RU Staying

Software Engineering
01:332:452
Report 3

By Group #11
Keya Patel
Zain Sayed
Mohammed Sapin
Purna Haque
Nga Man (Mandy) Cheng
Rameen Masood
Shilp Shah
Mathew Varghese
Thomas Tran
Eric Zhang

Github: https://github.com/mohammedsapin/RUStaying
Submission Date: April 14, 2019

Summary of Changes

- For the final submission, it was mainly about finishing up the features of the app completely. We made sure to have added system updates like forgot password, view pictures of the room, and made administrative pages for specific hotel staff to view their respective services (bellboy, valet, etc). We updated feedback to numerical data so we can later do analysis on the data. We fixed up on small tiny issues that make the application more consistent. We added many viewer versus guest restrictions while using the app; for example, the view rooms service page will only allow guests with account to book a room.
- We added a system update to visually represent the numerical data from the updated feedback forms to view the performance reviews of the services offered at the hotel. A credit card payment page was created so after guest select rooms and orders room service (food services) they will be direct to a payment page. Both food services page and administration page were implemented, so the guest will be given a menu of select options to choose from and be will finalized to payment and will be submitted as food service request.
- An inbox page was implement so that guest will be update on the status of their request they put in
- We implemented a pricing algorithm that is dependent on which days the guest books the room. For example, prices will be high during weekends,spring break and high demand months like December and Summer time months.
- Also, to allow more convenience, we generate unique digital rooms keys for each guest after they check in.
- Additionally, we decided to remove Use Case 10 because we felt it did not add too much value to our existing app.

Table of Contents

Individual Contribution Breakdown

Keya	10%	10%	10%	10%	10%	10%	10%
Zain	10%	10%	10%	10%	10%	10%	10%
Mohammed	10%	10%	10%	10%	10%	10%	10%
Purna	10%	10%	10%	10%	10%	10%	10%
Nga Man	10%	10%	10%	10%	10%	10%	10%
Rameen	10%	10%	10%	10%	10%	10%	10%
Shilp	10%	10%	10%	10%	10%	10%	10%
Mathew	10%	10%	10%	10%	10%	10%	10%
Eric	10%	10%	10%	10%	10%	10%	10%
Thomas	10%	10%	10%	10%	10%	10%	10%

Chapter 1

Customer Statement of Requirements

1.1 Problem Statement

Customer:

Tommy had a flight booked for American Airlines on Thursday. He wanted to enjoy a peaceful trip to Princeton. He booked a package deal on Expedia for flight and hotel. He had planned to stay at a local Hilton Hotel located in Princeton, NJ. Tommy was having a terrible time at the start of his vacation. His flight was delayed, TSA was giving him a hard time, and a baby was crying the entire flight. By the time he landed, he had waited 40 minutes for a runway to become available, another 40 minutes for his luggage to get to the carousel, and another 40 minutes for an Uber pool, which he shared with a slightly “eccentric” family.

By the time he got to the hotel, Tommy did not want to deal with another person, nor did he want to wait in line to be checked in. But to his disappointment, when he got to the hotel, the line was extremely long, held up by someone complaining at the front desk. After he finally got to the front of the line, checked in, and asked for a bell boy, he found himself again impatiently waiting for someone to come help him carry his heavy luggage up to his room. However, a bell boy was nowhere to be found, and Tommy had no choice but to wait in the lobby just for one to show up.

When he finally got to his room, all he wanted to do was rest from his terribly long day, but he could not even get into his room due to his key card malfunctioning. So he had to go back to the front desk and wait in line again for a new key. Once he finally got a new key and was able to enter his room he realized his sheets had not been changed, and the toilet wouldn't flush. He tried calling room service on the phone but the lines were busy, so again he had to go to the front desk and report the issues. However, the front desk told him that maintenance was busy, and that Tommy would have to wait for staff to check if another room was available.

As Tommy was hungry, he figured that he would let the hotel staff figure out his room situation while he grabbed a bite to eat. He walked over to the hotel restaurant, sat down at a table and relaxed in his chair. He figured that after a meal he would be in better spirits. However, when the waiter handed him his menu, to his horror he discovered that the hotel had no vegan options. Tommy, a devout believer in veganism, found his efforts to relax again thwarted by a lack of information readily available for him, and attempted to leave the hotel to find outside dining options. But as soon as he stepped outside, he was immediately hit with a wall of torrential rainfall. Now fuming, he decided to check on the room status, and to his pleasant surprise, the hotel had been able to quickly move his belongings to a new room. Entering, he checked the toilet, sink, and bed for any signs that his new room was dysfunctional. Satisfied, he laid in his bed, ready to finally get a good night's rest after such as stressful day. However, he quickly discovered that sleeping would not be an option, as the hot and humid summer night caused a profusion of sweat to begin forming all over his body. He quickly reached for the air conditioning remote, and turned it on to its lowest setting. To his dismay, the AC began sputtering hot, foul smelling air into his room. At this point, Tommy was ready to cut his vacation short and go home. After again going to the front desk and finding a new room, Tommy finally fell into a nightmare riddled sleep full of bathroom malfunctions and dirty bed sheets. When he awoke, he felt surprisingly refreshed with a clear purpose. He gathered all of his belongings and promptly checked out of the hotel. He had decided to go back home after all. On his way out, he asked if the hotel had any customer feedback forms he could fill out. After being told no, he happily yelled at the staff for an hour before departing for his flight home.

From Tommy's experience we can see that many issues can arise in a hotel. This could be do to poor hotel management, influx of many guests during vacation weekends, and many other things. The hotel was able to provide Tommy with solutions for each of his problems, however it was very inconvenient and took a long time. Tommy would have rather had his solutions addressed or solved at a timely manner. With our application we are trying to solve these issues and help people like Tommy have a better guest experience at hotels. Through the app Tommy can check in beforehand and receive a digital copy of his key on his phone. Before he enters the hotel he can even request for his room to have clean sheets, working appliances, etc. All of this saves Tommy time from waiting on the phone or going down to the front desk each time to solve an issue. Through the app we are enhancing the customers ability to have self service and do things on their own. Hotels will still operate the same way and guests can choose not to use the app and talk directly to the hotel staff. We are targeting a group of people that are more tech savvy and are part of the revolutionizing era of self-service technology. A typical guest through the app can request a car, report how many luggages he has, request amenities for his room, preview menus, and even order food. At the end of a users stay the app asks for feedback and ways to improve service.

Hotel Staff:

Kenny, the hotel manager, is tired from a long, endless night of checking people into the Varghese Hotel. As he stands at the desk, typing furiously away trying to check-in people waiting, he is getting more stressed. Kenny tries his best to meet every guest with a smile and a friendly greeting, however, it becomes difficult when guests come in exhausted and irritated from their travels. Currently, it's 5 PM on a Friday night and there are 15 guests waiting impatiently to get checked-in. Kenny, thinking to himself, wishes that there was a way all of these problems could go away. If only there was an efficient way to check people in without angry customers yelling at him. As Kenny works tirelessly to make sure all customers are shown to their rooms, he realizes there are not enough bellboys for guests' luggage. The guests, realizing there is no bellboy, furiously just takes their luggage to their rooms themselves. Their stay at the hotel is not starting on a good note. As Kenny takes a quick moment to compose himself, his break is hurried with ring of the front desk phone. The complaints begin to pour in. Guests are filing maintenance requests, bedding changes, issues with the room keys, and much more. Although I have my hotel managing staff to help me, all the complaints are starting to pile up and guests are growing impatient. We are confident we will resolve all the issues, but not in a timely manner. On top of this, the technology we use at the hotel has many glitches. We try our best to keep track of room usage and just general monitoring, however our tech is outdated and slow. We need to make advancements and improve the way we manage the hotel.

App Solution (Manager/Staff):

- Manager can view assignments of digital keys for security purposes

- Bellboys will get a push notification when service is needed
- Manager can see which rooms are occupied and pull up the information of guest staying in that room, and the requests the guest may have listed
- Maintenance staff will be notified for requests

With the app (Solution):

- Put in luggage information through the app to make sure bellboy service is on time
- Digital Key to avoid problems with the physical key plus skip the LONG lines
- Put in urgent maintenance request (broken toilet, phone, AC), no need to call
- Ask for room cleaning (Dirty bed sheets) , no need to call
- Check for restaurant hours
- Check the restaurant menu through the app and include dietary restrictions
- Eliminate wait times through check in with the app, and requests made in the app
- Feedback form

Although Tommy was having a horrible course of events, these could have all been avoided through the use of the hotel app, RU Staying. Through the app, Tommy had the option to request a bellboy upon his arrival and even say how many luggages he will have when he does arrive. In addition, the digital key that is available through the app would have allowed Tommy to avoid any malfunctions with a physical card key. These are just a few of many features that the app will be able to provide to guest who are seeking a more enjoyable and convenient stay.

1.2 Glossary of Terms

Bellboy - A staff member who helps guests in moving their luggage to their rooms, and provides any additional assistance that may be asked of them.

Maid - A staff member whose primary responsibility is to clean the guests' rooms.

Maintenance request - Process by which a guest can request assistance when dealing with a technical or mechanical issue in their room

Reservation - Process by which a guest will reserve an available room and pick out other amenities or logistics

Food Service - Process by which guest will order food up to their rooms.

Check-In - Process by which a guest will confirm their booking information and payment methods and be allowed to move into their hotel room to begin their stay.

Digital Room Key - A digital card that, after booking a room, the guest will receive on their phones after checking in. It will allow them access to their rooms and other guest-only areas in the hotel such as the pool or gym.

Physical Room Key - A physical card version of the digital room key- it will allow guests access to their rooms and other guest-only areas in the hotel such as the pool or gym.

Creating an account - A process by which a guest creates a username and password to become associated with the hotel, and allow their information to be quickly accessed by hotel staff.

Guest - A person who has booked a room, checked in, and is now living in the hotel within his reserved duration

Valet Parking - A process by which a bellboy or another staff member will be given a guest's keys and drive the guest's vehicle to the hotels parking garage. The guest can also request for their car to be driven back to the front entrance if they wish to leave.

Feedback/customer satisfaction - Through a form we will gauge how much customers either enjoyed or did not enjoyed living at our hotel. This can be measured by any complaints or complements they may have, as well as any additional comments that we will read.

Luggage - Any amount of baggage that guests bring with them to keep during their stay.

Bedding - Additional room items such as bed sheets, pillows, blankets, towels, etc.

Chapter 2

Functional System Requirements

2.1 User Stories

REQ	Priority Weight	Description
REQ - 1	2	System will allow users to create an account and register with email and password
REQ - 2	2	System will allow users to login to their account with a unique username and password
REQ - 3	6	System will allow users to check-in for their room
REQ - 4	6	System will allow users to make reservations
REQ - 5	4	System will allow users to call room service.
REQ - 6	8	System will keep track of available / unavailable rooms
REQ - 7	5	System will allow users to make maintenance requests
REQ - 8	3	System will verify user account by checking with the database
REQ - 9	2	System will allow users to input how many bags they have in order to call the bellboy
REQ - 10	7	System will provide the user with a digital room key.
REQ - 11	1	System will provide the user with additional hotel information

		and FAQ.
REQ -12	3	System will allow guests to use the car service.
REQ - 13	10	System will be able to keep track of room usage and general hotel activity
REQ - 14	1	System will ask for feedback from the guest upon checkout
REQ - 15	3	System will allow users to easily communicate with staff
REQ - 16	6	System will allow users to check-out of their room
REQ - 17	1	System will provide an automated concierge service to answer questions
REQ - 18	1	System will allow user to reset their password if they forget it
REQ - 19	2	System will allow user to view picture of rooms, rooms availables and prices prior to login
REQ - 20	1	System will report numerical feedback data (ratings, service usage, etc.)
REQ - 21	1	There will be one manager account for each respective hotel staff member.
REQ - 22	1	Appropriate service requests will be sent to the respective managerial accounts (one account does not receive all accounts)

2.2 Non-Functional System Requirements

REQ	Priority	Description

	Weight	
REQ - 23	6	The app will have a simple, responsive user interface (see details below about more specific requirements)
REQ - 24	1	The system should allow more than one user to access and use the application
REQ - 25	1	The system will use Firebase for data maintenance
REQ - 26	2	The system should provide error notifications to let the user know there is an issue

2.2.1 Functionality

The goal of this app is to become widely used by all the guests that stay at the hotel. Overtime, we expect a growing user base because more people will visit the hotel and revisit because of the convenience we provide through the app. To handle the growing user base, we will be storing our information using Firebase database. Since most of the data we are storing is user information and the use of the hotel services, we needed a realtime database to share and sync data across all devices. Powerful data sorting and analysis tools are not needed so we decided against Postgresql. Since the app is on Android, it is already portable and usable by most devices in the world. For security, we will be using the built in password and username encryption provided by Firebase.

2.2.2 Usability

As mentioned, the app is being built on the Android platform so it is very accessible by many users. To have a modern app design that is aesthetic and responsive, we will be following

Google's Material Design tools. This will save us a lot of time when creating the user interface because all the tools are built and we only have to implement them.

2.2.3 Reliability

We will be thoroughly debugging and testing our application to ensure that it does not crash at any point during use. Throughout our multiple builds, we will be continually updating our application to fix any issues that arise in runtime. This will help cut down on the possibility of failure if and when an actual customer is using it. Additionally, since our data is stored on Firebase, we are almost guaranteed that data will not get lost or corrupt. We will also be making sure that customers will not have access to the managerial aspect of the app and vice versa to prevent any security risks or issues.

2.2.4 Performance

For the purposes of this app, we will not be managing a very large set of data, so the performance that Firebase provides is acceptable. The app is more focused on functionality and Android Studio is a reliable application that allows us to scale the app easily.

2.2.5 Sustainability

When implementing the requirements of the app, we will heavily use the features of OOP to maintain modularity and scalability. Also, we will use the Android Studio debugging tools to ensure an error-free experience for the user. Of course, errors can still occur so we will provide notifications in the app of the errors that occur. This way, while testing the app, we can see the type of errors and where they occur to try to fix them.

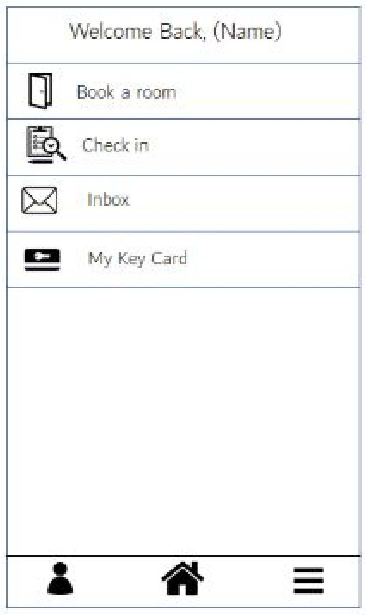
2.3 On-Screen Appearance Requirements

The on-screen requirements are specific to the user interface and were designed to provide a simple and convenient experience. We will be splitting up the user interface in two main parts, the first part is a page that provides the necessary information and the second part is a navigation bar that is consistent across all pages. This is a common modern design for many applications as it is simple and allows the user to navigate the app easily.

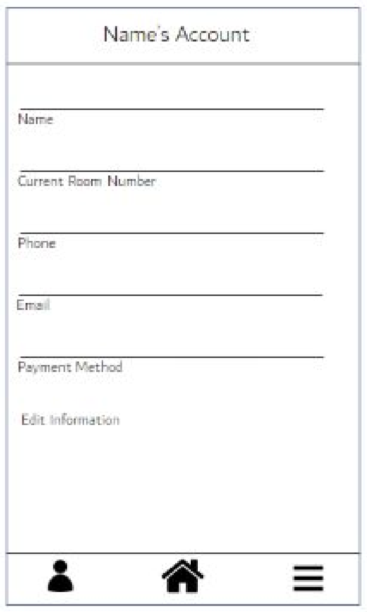
REQ -	Priority Weight	Description
REQ - 27	1	The app will have a navigation bar across the bottom to switch easily between the main features (ex: Home, Check-in, Guest Services, User Profile etc…).
REQ - 28	1	The Home button on the navigation bar will lead to a page where the user can explore more about the hotel, nearby points of interests, and FAQ.
REQ - 29	1	The User Profile tab will provide the users personal information and details about their hotel stay.
REQ - 30	1	Another tab will allow the user to Check-In or Make a Reservation. Upon clicking on these, it will lead to another page where the user can actually perform the action.
REQ - 31	1	We have a lot of services that the user can request so we will have one tab that lists all the services and from there the user can select one (see UI diagrams below).
REQ - 32	3	Within the Request Hotel Service tab, some of the services listed are: Room Service, Valet & Travel Service, Call Bell Boy, Room Maintenance, and the Feedback Form. This is the main tab where the user can access all the services.

2.3.1 App User Interface Templates

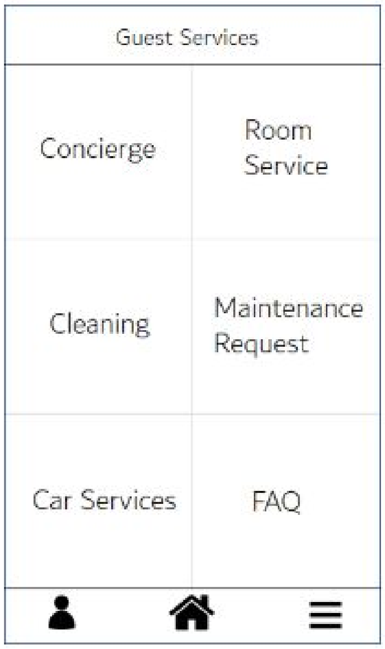
Home Page



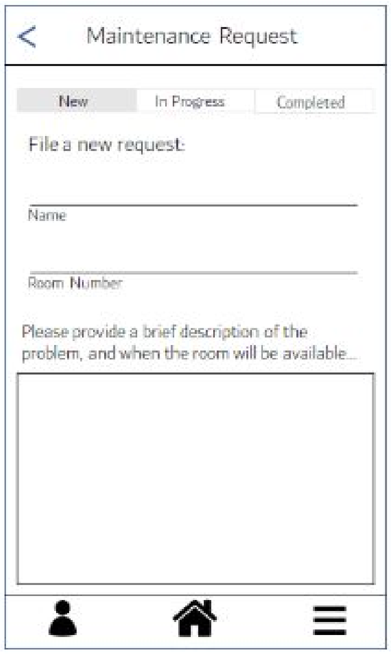
Account Page



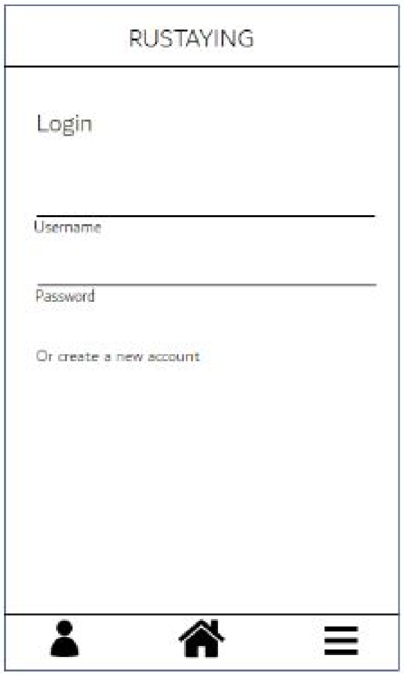
Guest Services ☰

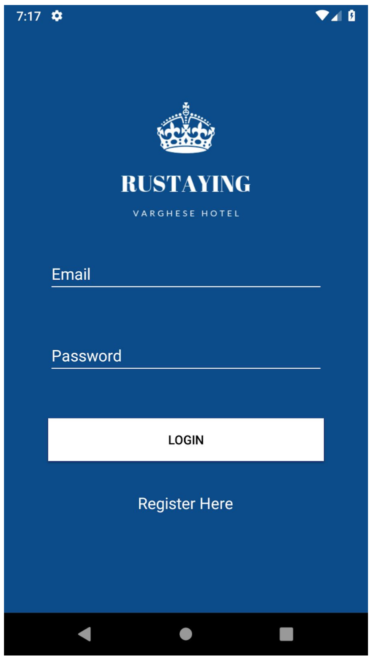


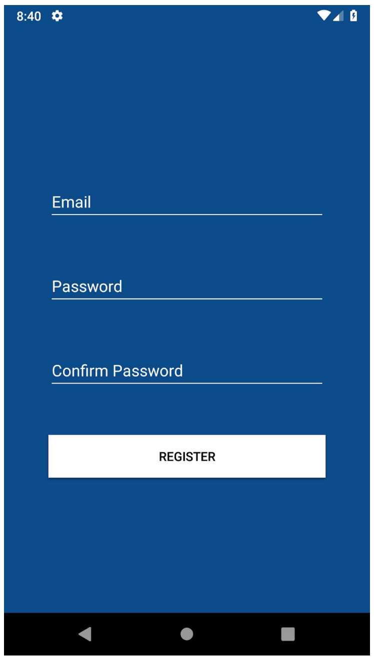
Maintenance Request Page (from Guest Services)

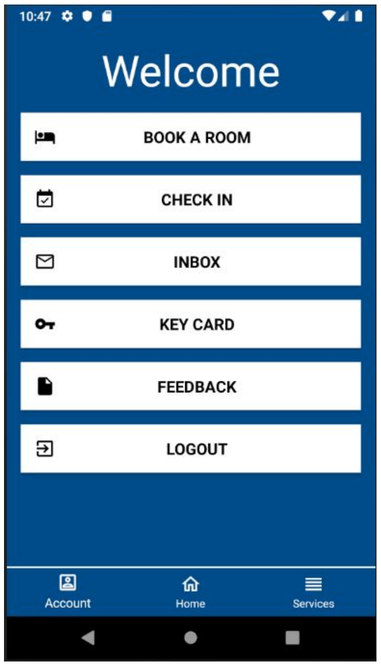

Maintenance Request
New
In Progress
Completed
File a new request:
Name
Room Number
Please provide a brief description of the problem, and when the room will be available...

Login Page









These three images are the final design that we are planning to implement for our application based on the mock-ups we created. The first image displays the page a user sees when he first opens our app. If a user does not have an account they can click Register Here and will be navigated to the another page as displayed by the second image. When a user successfully logs in to the app the first page they see is the welcome page as displayed by the third image. From here a user will be able to access all the services our app provides for them.

Chapter 3

Functional Requirements Specification

3.1 Stakeholders

The main stakeholders of this app are hotel owners and managers. The app is commissioned by the owners of the hotel in an attempt to expand the hotel brand and earn more profit. With our design, this app can significantly improve the efficiency of the hotel services, which in turn increases the number of guests, therefore increasing profit. As investors, the hotel owners are certainly interested in the growth and success of the app. The hotel managers are also stakeholders because they control the day-to-day activities of the hotel and it is up to them to use the data our app provides to efficiently schedule staff and hotel services. Also, all the users and guests of the app are stakeholders too because through their support and feedback, we can continue to improve the features and user interface. And finally, all the developers of the RUStaying app since we are investing a lot of time and effort into the creation of the system. As developers, our role is to ensure the success of the app and continue to make improvements based on the feedback from the other stakeholders.

3.2 Actors and Goals

Key
I → Initiating
P → Participating

Actors	Goals
Guests - I	Will be able to login and send requests to Hotel Staff as needed
Admins/Manager- I	Primary initiating goal is to be able to monitor customer interaction with hotel services. Will be able to log in and view requests as they need.
Hotel Staff (includes Manager) - P	Guests will be sending multiple requests for various services throughout the day. Staff receives and will act accordingly upon request. Restaurant will view orders, maintenance can view forms,etc..
Database - P	Will store all essential information from guests and general hotel information. This will be an efficient way to organize and access necessary info on guests.

3.3 Use Cases

The RUStaying app has 2 main users, the guest and admin. Each user has their own separate use cases

1. Register (guest) - to register an account on the application

2. Login (Guest & admin) - to log into a user account

3. Logout (Guest & admin) - to log out of a user account

4. Make a reservation (Guest) - to make a reservation for a room
5. Check In (Guest) - to check into the hotel
6. Room Service (Guest) - to call for room service
7. Check availability of Rooms (Admin & Guest) - to check which rooms are in use or vacant
8. Review Guest Data (Admin) - to check the information of guests, given a specific room
9. Request Status (Admin) - to be able to change or mark a guest request “in progress” or “granted”
10. Concierge (Guest) - access front desk information through the app
11. Maintenance (Guest) - to call for maintenance
12. Food (Guest) - order food
13. Valet & Travel (Guest) - to call for a car to the airport or wherever the guest wants to go, or call for a valet to park the guest’s car
14. Feedback (Guest) - to allow the guest to provide feedback after their stay
15. Hotel Information (Guest) - to give guests a visual of the restaurant menu, gym hours, pool hours, and spa hours
16. Request Bellboy Services (Guest) - to call a bellboy to get guest’s luggage
17. Checkout (Guest) - to check out of the hotel
18. Forgot password (Guest) - ability to reset the password if a guest forgets
19. Report feedback data (Admin) - Manager accounts can see feedback data submitted by guests
20. Preset Manager accounts (Admin) - Each type of hotel staff will have a manager account for handle service requests
21. Send service requests to appropriate admin account (Admin) - The service requests will be sent to the appropriate manager account to be handled by staff members.

3.3.1 Use Case Diagram

3.3.2 Fully Dressed Use Cases

Use Case 1: Register an account
Related Requirements: REQ-1
Initiating Actors: Application User
Actor's Goal: To create an account on RUStaying app
Participating Actors: Database (keeping track of past emails used)
Preconditions: 1. The email the Guest is using to register has not been used previously in our system 2. The email and password meet certain criteria for security
Postconditions: User has an account to access the features of the app
Flow of events for main success scenarios: 1. User opens app and on login page chooses "Register an Account" 2. Fills out the required account information 3. Check for valid email and password 4. Store guest data in database 5. New account is successfully created

Use Case 2: Login to an account
Related Requirements: REQ-1, REQ-2, REQ-22
Initiating Actors: Application User
Actor's Goal: To log into an existing account created on RUStaying App
Participating Actors: Database
Preconditions: 1. User has an already existing account
Postconditions: User is able to back into his/her account
Flow of events for main success scenarios: 1. The user has opened the app and on the login page fills in their email and password 2. If email/password are incorrect, system notifies user

3. The user successfully logins into the desired account

Use Case 3: Logout

Related Requirements: REQ-1, REQ-2, REQ-3

Initiating Actors: Application User

Actor’s Goal: To exit their account in the app

Participating Actors: Database

Preconditions:
1. The user already has an existing account
2. The user is currently logged into the system

Postconditions:

Flow of events for main success scenarios:
1. The user has logged in and initiates logout function
2. Database confirms user account
3. Database marks user account as logged out

Use Case 4: Make a reservation

Related Requirements: REQ-1, REQ-2, REQ-4, REQ-6, REQ-8, REQ-30

Initiating Actors: Application User/Guest

Actor’s Goal: To book a room for the duration of their choosing and/or luxury of their choice

Participating Actors: Database (to check for accommodations)

Preconditions:
1. Guest must have created an account with our app
2. Guest has successfully logged in

Postconditions: Guest will receive confirmation via app

Flow of events for main success scenarios:
1. User has logged in and selected the “Make a Reservation” option
2. User fills out details of what type of room they are looking for (price, number of people, size of room, etc..)
3. System will relay information and gather available rooms and present options to user

4. User will choose an option
5. System has successfully booked the room for user

Use Case 5: Check In

Related Requirements: REQ-3, REQ-10, REQ-23

Initiating Actors: User

Actor’s Goal: User would like to check in for their room upon arrival

Participating Actors: Hotel Staff, database (to check for reservation)

Preconditions: User has created an account and made a reservation

Postconditions: User received digital room key and can access services.

Flow of events for main success scenarios:

1. User has logged in and selects the “Check In” option on arrival
2. System verifies user and provides user with a digital room key.

Use Case 6: Room Service

Related Requirements:REQ-2, REQ-3, REQ-5

Initiating Actors: Application User/ Guest

Actor’s Goal: To be able to call room service for a variety of purposes (Clean the room, replace toiletries, replace bed sheets, etc.)

Participating Actors: Room Staff, Maids

Preconditions:

1. Guest has been checked-in before requesting any service

Postconditions: Guest will be notified when request has been sent

Flow of events for main success scenarios:

1. User has logged in and has selected “Request Room Service”
2. System will confirm check in status of guest and return with a list of services available to guest
3. User will choose an option along with the timing of when it is needed
4. System will confirm and send notification to team responsible for service

Use Case 7: Check Availability of Rooms
Related Requirements: REQ - 6, REQ - 13, REQ - 19
Initiating Actors: Guest, Admin
Actor’s Goal: User would like to check the availability of the rooms
Participating Actors: Guest Admin
Preconditions: User opens the app (No longer need to login to view available rooms)
Postconditions: App displays availability of rooms
Flow of events for main success scenarios: 1. User has logged in and selects “Make a Reservation” 2. System pulls from database availability of rooms 3. User is able to see which rooms are vacant

Use Case 8: Review Guest / Service Data
Related Requirements: REQ-13
Initiating Actors: Admin
Actor’s Goal: To view the guest data, room availability and usage of hotel services
Participating Actors: Database
Preconditions: 1. Accounts created by guests 2. Guests request hotel services from app
Postconditions: 1. Display guests data (ex: Number of guests in hotel) 2. A breakdown of how much each service is used 3. List of rooms available / occupied
Flow of events for main success scenarios: *This use case is for all data storage so it <<includes>> all other use cases* 1. Guests continue to use services through the app 2. Data is kept track of and it sent to the database to store 3. The data is collected from the database frequently

4. The data is presented to the Admin in a user friendly format

Use Case 9: Request Status
Related Requirements: REQ-5, REQ-7, REQ-9, REQ-12, REQ-13, REQ-24
Initiating Actors: Admin
Actor's Goal: To be able to change or mark a guest request "in progress" or "granted"
Participating Actors: Guest, Hotel Staff, Database of Hotel Staff Activities
Preconditions: 1. Guest must have made a request for some type of service, see REQ-5, REQ-7, REQ-9, and REQ-12
Postconditions: 1. Guest request will be either labeled "in progress" or "granted"
Flow of events for main success scenarios: 1. User has logged in and makes a request from one of those offered in the app 2. System receives request and labels the request as "in progress" 3. User reports to the system when service has been completed 4. System marks requests as "granted"

Use case 10 is longer part of our project. We decided to eliminate this use case because we felt it did not add too much value to our existing app. After Demo 1, we realized there are many other things we need to add to our app for better usability and functionality, so Use Case 10 is no longer a priority.

Use Case 11: Maintenance
Related Requirements: REQ-2, REQ-7
Initiating Actors: Guest
Actor's Goal: To submit a maintenance request into the app and for a hotel worker to solve the issue
Participating Actors: System, Hotel Staff, Maintenance Worker
Preconditions: 1. Guest is logged in

2. Guest needs to submit maintenance request
Postconditions: 1. Guest will have maintenance requests fulfilled 2. Confirmation message will pop up after fulfilled
Flow of events for main success scenarios: 1. User has logged in and selects “Maintenance Requests” 2. Database pulls options for Maintenance Requests 3. User will be able to select from the options presented 4. App will submit option into database 5. Database will send confirmation message that request was successfully submitted

Use Case 12: Valet & Travel Services
Related Requirements: REQ 12, REQ 25
Initiating Actors: Guest
Actor’s Goal: To call for a car to the airport or wherever the guest wants to go, or call for a valet to park the guest’s car
Participating Actors: Hotel Staff, Car
Preconditions: 1. Guest is logged in 2. Guest has checked in
Postconditions: 1. Guest will have their requested car waiting outside 2. Guest will be notified once the car is called and ready
Flow of events for main success scenarios: 1. User has logged in and selects “Guest Services” 2. User can select “Valet & Travel” and choose to call a car 3. Car Services will indicate that they have arrived for pick up 4. System will notify User that car has has arrived and is ready

Use Case 13: Feedback
Related Requirements: REQ-14
Initiating Actors: Guest

Actor's Goal: To allow the user to input feedback upon check out
Participating Actors: Database
Preconditions: 1. User has checked out of the hotel
Postconditions: 1. Check out confirmation will pop up 2. Feedback form will pop up
Flow of events for main success scenarios: 1. User has logged in and selects "Check out" option at the end of their stay 2. System will output a check out confirmation 3. System will output a feedback form from Database 4. User will fill out or dismiss feedback form and submit it into the Database 5. System will output a submission confirmation

Use Case 14: Hotel Information
Related Requirements: REQ-11
Initiating Actors: Application User
Actor's Goal: To view restaurant menus, gym hours, pool hours, spa hours at the user's convenience, etc...
Participating Actors: Database
Preconditions: 1. Guest is logged in
Postconditions: 1. Guest will be informed of hotel information
Flow of events for main success scenarios: 1. User has logged in and selects "View Hotel Information" 2. System will display hotel information

Use Case 15: Bellboy Services
Related Requirements: REQ-9 REQ-25

Initiating Actors: Guest
Actor’s Goal: get their luggage moved to their room
Participating Actors: Hotel Staff
Preconditions: 1. Guest is logged in 2. Guest has checked in
Postconditions: 1. Guest will have their luggage taken up to their room 2. Guest will be notified once luggage has been taken up
Flow of events for main success scenarios: 1. User has logged in and selects “Guest Services” 2. User can select bellboy and choose to move luggage to room 3. Bellboy will indicate that they have finished moving luggage 4. System will notify User that luggage has been taken up to their room

Use Case 16: Checkout
Related Requirements: REQ-16
Initiating Actors: Application User
Actor’s Goal: To check out of the hotel through the app
Participating Actors: Hotel Staff, Database
Preconditions: 1. Guest has finished their stay at the hotel
Postconditions: 1. Guest will be able to write a feedback report
Flow of events for main success scenarios: 1. User has logged in and selects “Check out” after finishing their stay at the hotel 2. User is given the option to have a bellboy help move luggages to the main lobby 3. Database is notified that room service is needed 4. System accesses database to update that the room is available

Use Case 17: Forgot Password

Related Requirements: REQ - 18
Initiating Actors: Guest
Actor's Goal: To reset their password
Participating Actors: Database
Preconditions: 1. Guest already has an account
Postconditions: 1. Guest is able to login with their new password
Flow of events for main success scenarios: 1. Guest clicks on forgot password 2. Guest will receive a link to reset their password 3. Guest is able to login with the new password

Use Case 18: Report Feedback Data
Related Requirements: REQ - 20
Initiating Actors: Admin
Actor's Goal: To report feedback data to improve guest experience
Participating Actors: Database
Preconditions: 1. Admin has access to all feedback data
Postconditions: 1. Admin is able to see a visual representation of the feedback data
Flow of events for main success scenarios: 1. Admin clicks on a button to report real time data

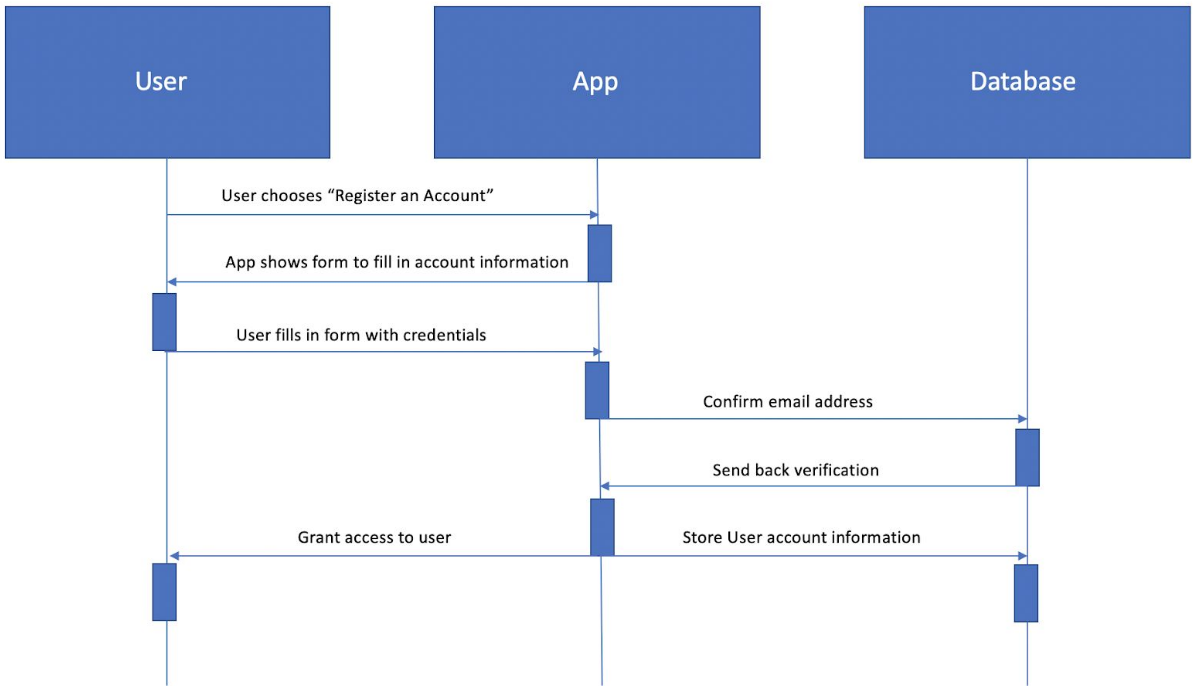
Use Case 19: Preset manager accounts for hotel staff
Related Requirements: REQ - 21
Initiating Actors: Manager / Hotel Staff
Actor's Goal: To view service requests from guests

Participating Actors: Hotel Staff, Database
Preconditions: 1. Hotel staff or manager opens app
Postconditions: 1. Each type of hotel service has its own admin account (ex: Bellboys have an admin account)
Flow of events for main success scenarios: 1. Preset emails for each hotel staff 2. Hotel staff logins with correct credentials

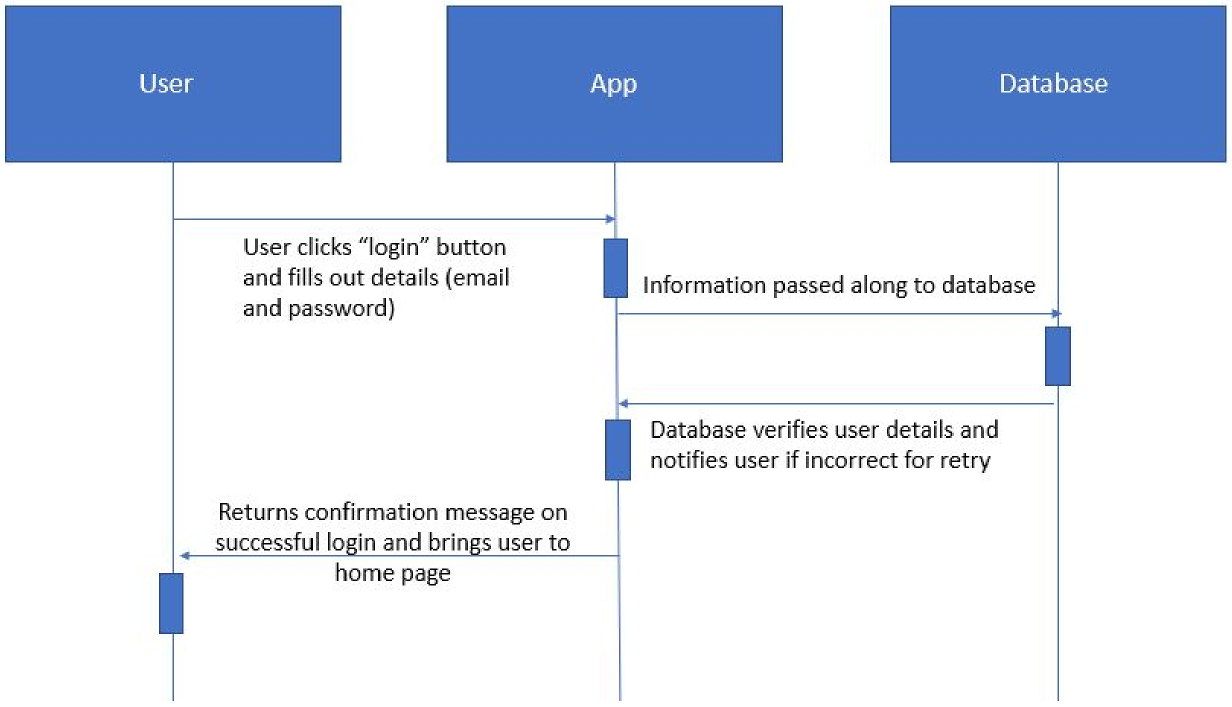
Use Case 20: Send service requests to appropriate manager account
Related Requirements: REQ - 22
Initiating Actors: Manager / Hotel Staff
Actor’s Goal: To view service requests from guests
Participating Actors: Hotel Staff, Database
Preconditions: 1. Service requests made my guests
Postconditions: 1. The service requests gets sent to the correct hotel staff account (ex: Bellboy request only shows up on bellboy account)
Flow of events for main success scenarios: 1. Hotel staff is logged into their account 2. Service requested by guest 3. Service request shows up on staff account

3.3.3 System Sequence Diagrams

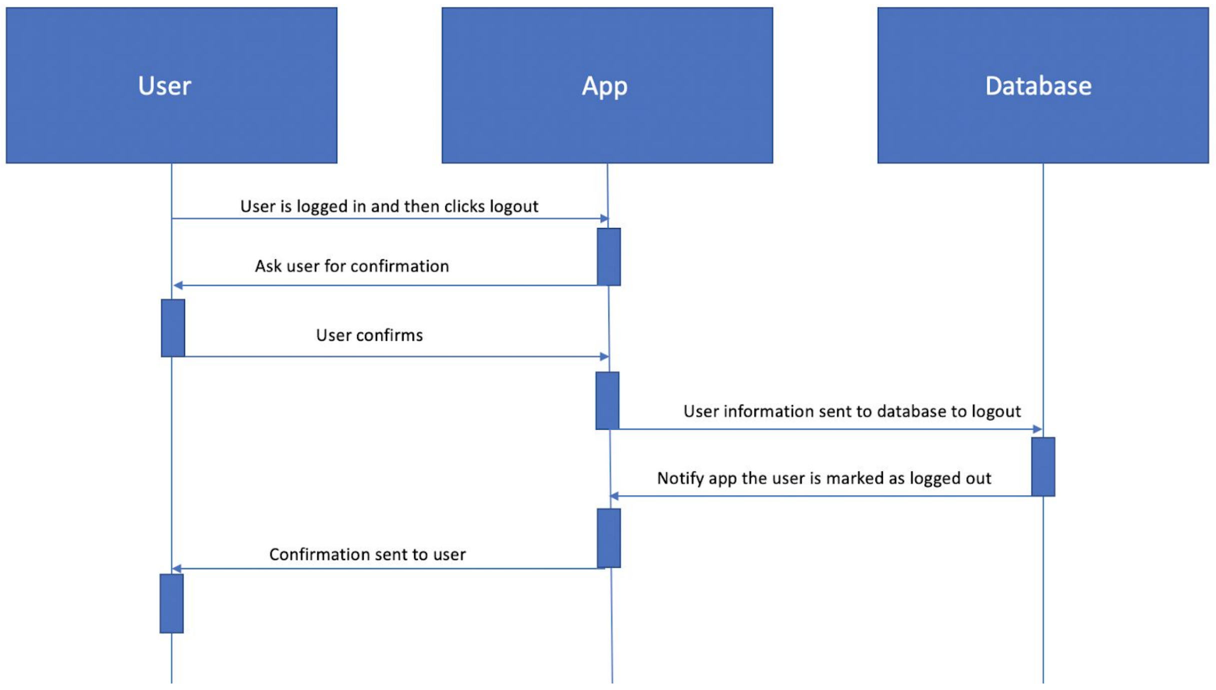
Use Case 1: Register an Account


User
App
Database
User chooses "Register an Account"
App shows form to fill in account information
User fills in form with credentials
Confirm email address
Send back verification
Grant access to user
Store User account information

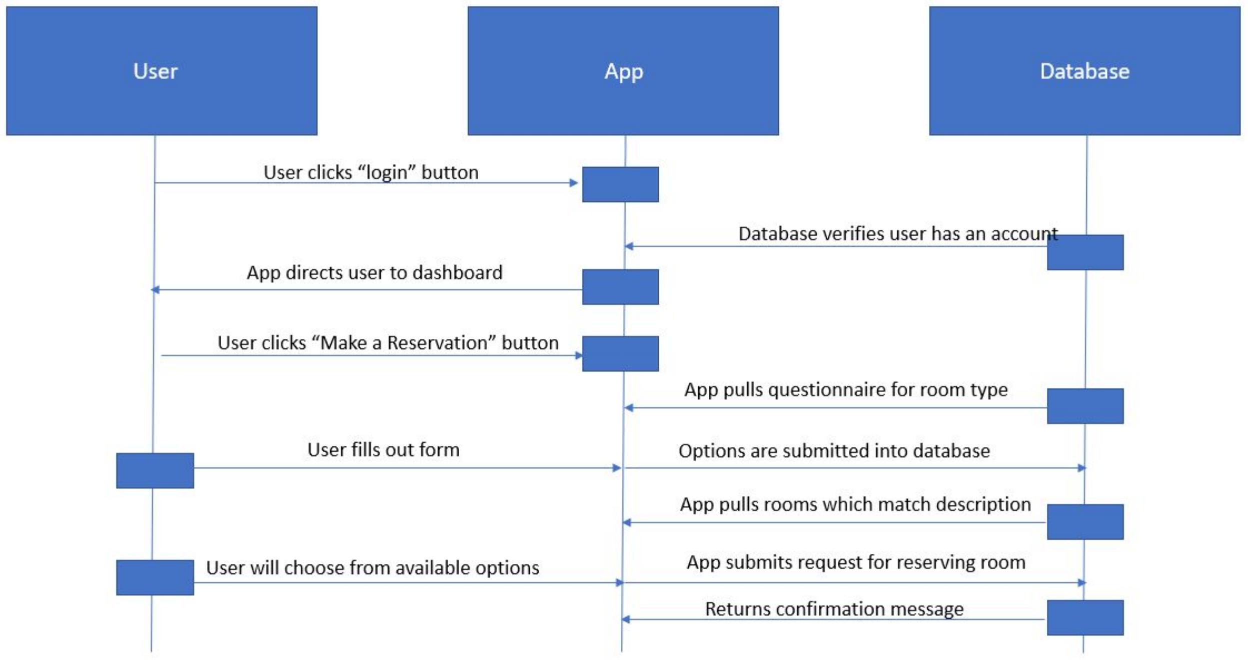
Use Case 2: Login


User
App
Database
User clicks "login" button and fills out details (email and password)
Information passed along to database
Database verifies user details and notifies user if incorrect for retry
Returns confirmation message on successful login and brings user to home page

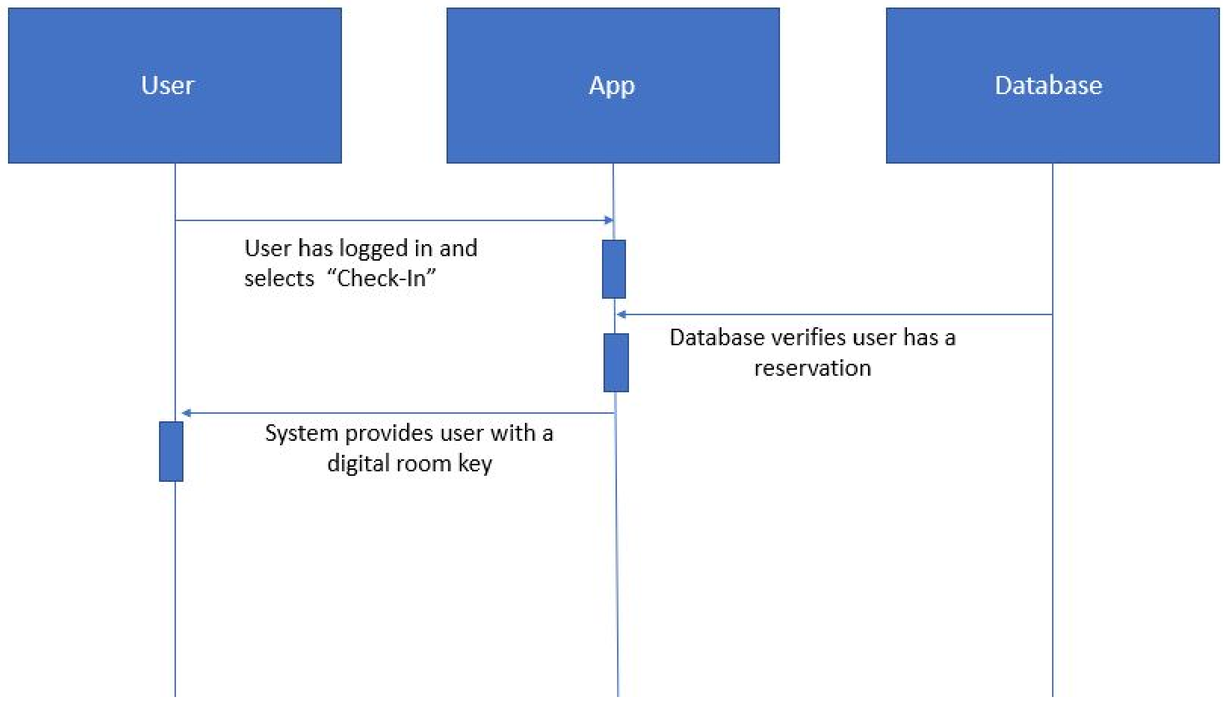
Use Case 3: Logout


User
App
Database
User is logged in and then clicks logout
Ask user for confirmation
User confirms
User information sent to database to logout
Notify app the user is marked as logged out
Confirmation sent to user

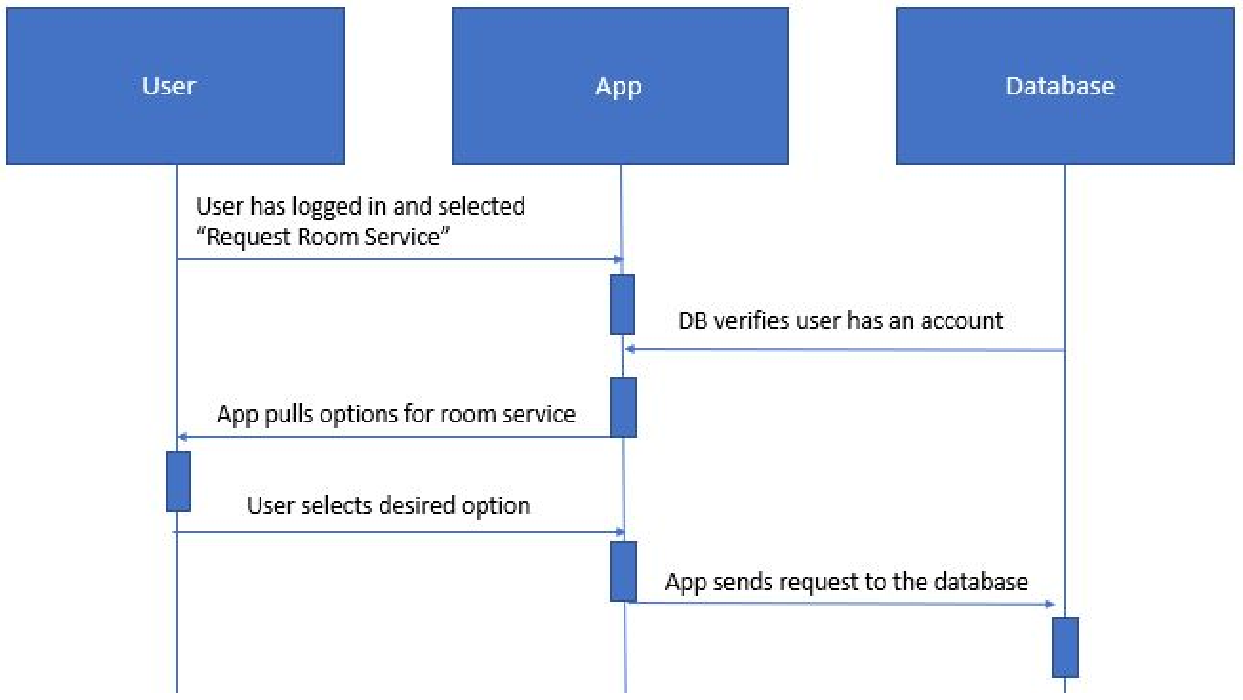
Use Case 4: Make a Reservation


User
App
Database
User clicks "login" button
Database verifies user has an account
App directs user to dashboard
User clicks "Make a Reservation" button
App pulls questionnaire for room type
User fills out form
Options are submitted into database
App pulls rooms which match description
User will choose from available options
App submits request for reserving room
Returns confirmation message

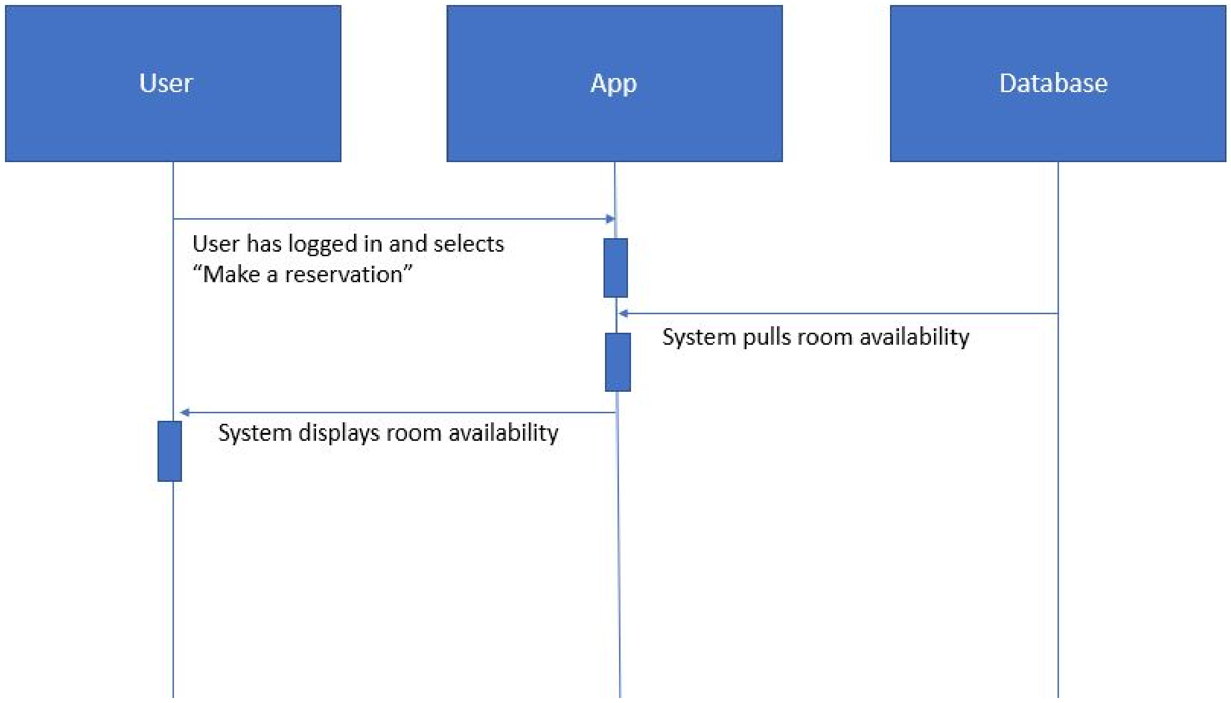
Use Case 5: Check-In


User
App
Database
User has logged in and selects "Check-In"
Database verifies user has a reservation
System provides user with a digital room key

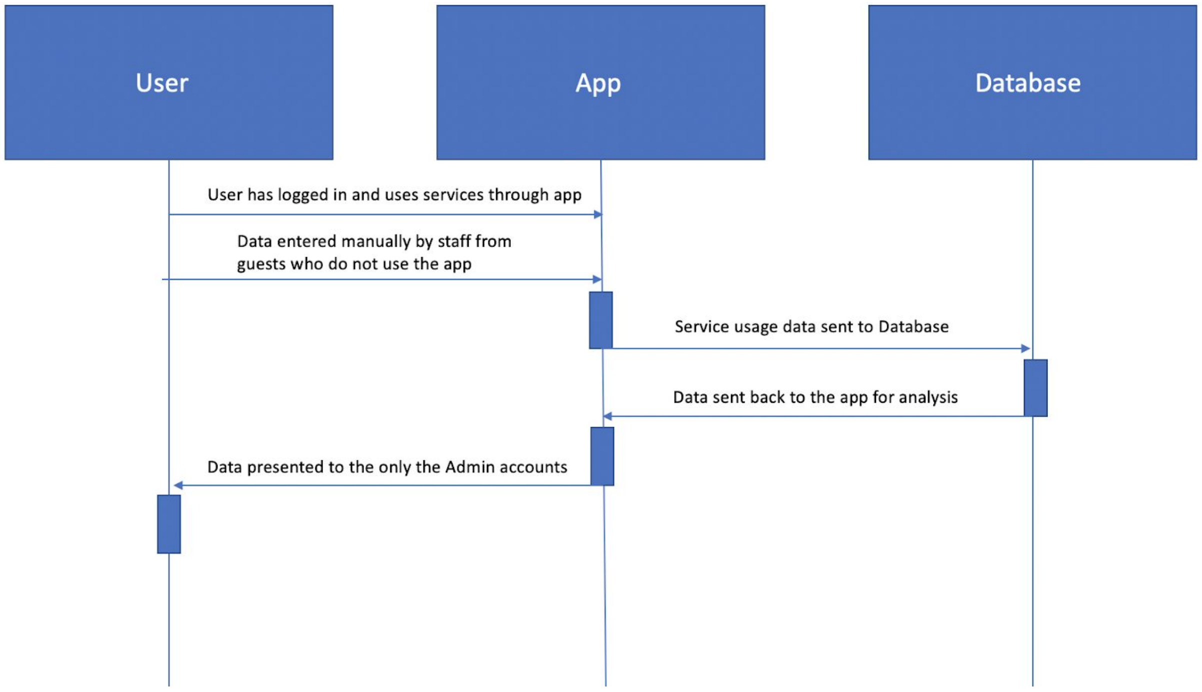
Use Case 6: Room Service


User
App
Database
User has logged in and selected "Request Room Service"
DB verifies user has an account
App pulls options for room service
User selects desired option
App sends request to the database

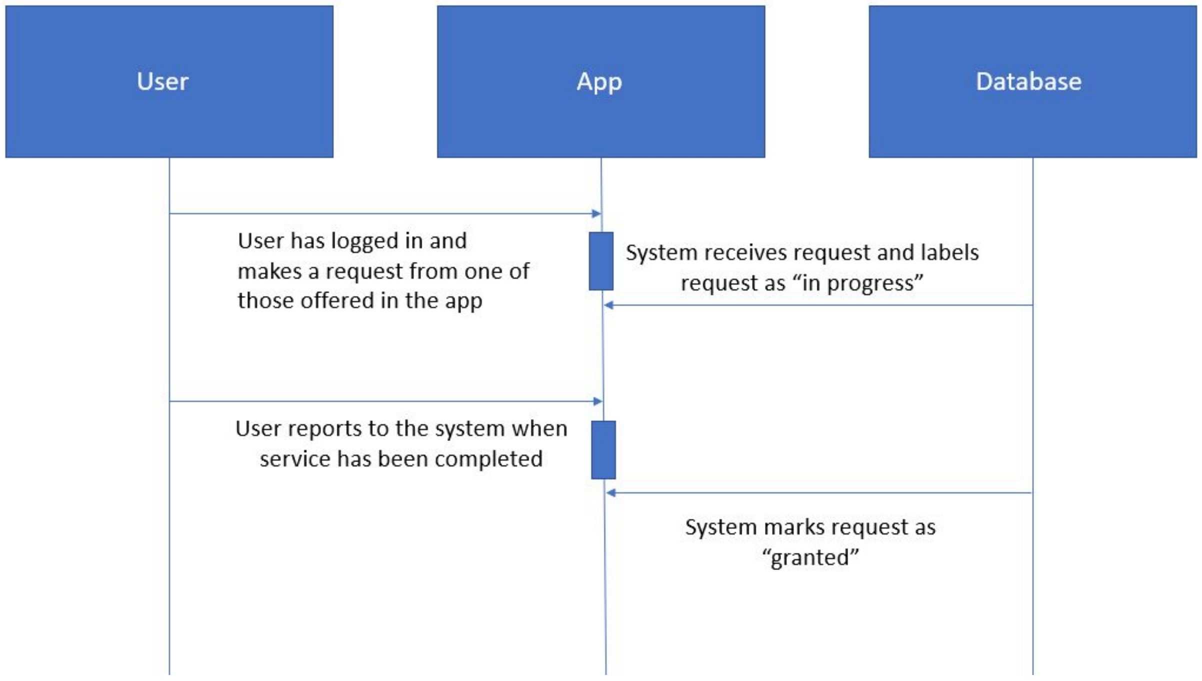
Use Case 7: Check Availability of Rooms


User
App
Database
User has logged in and selects "Make a reservation"
System pulls room availability
System displays room availability

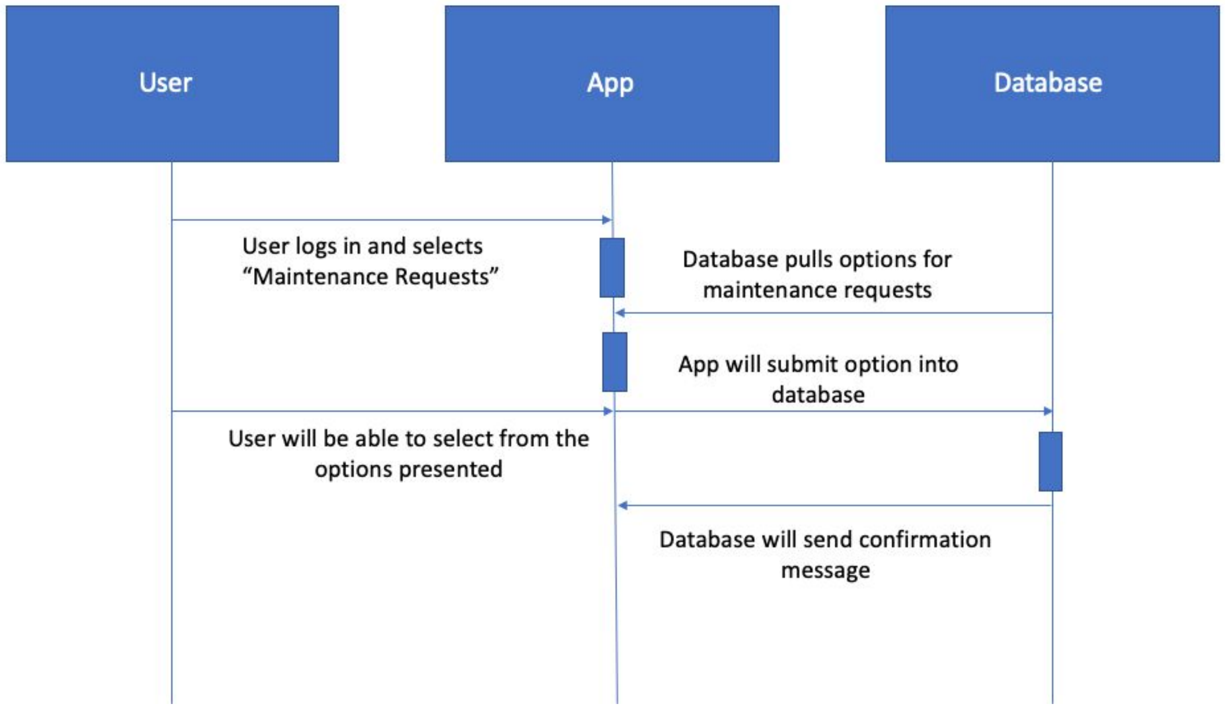
Use Case 8: Review Guest / Service Data


User
App
Database
User has logged in and uses services through app
Data entered manually by staff from guests who do not use the app
Service usage data sent to Database
Data sent back to the app for analysis
Data presented to the only the Admin accounts

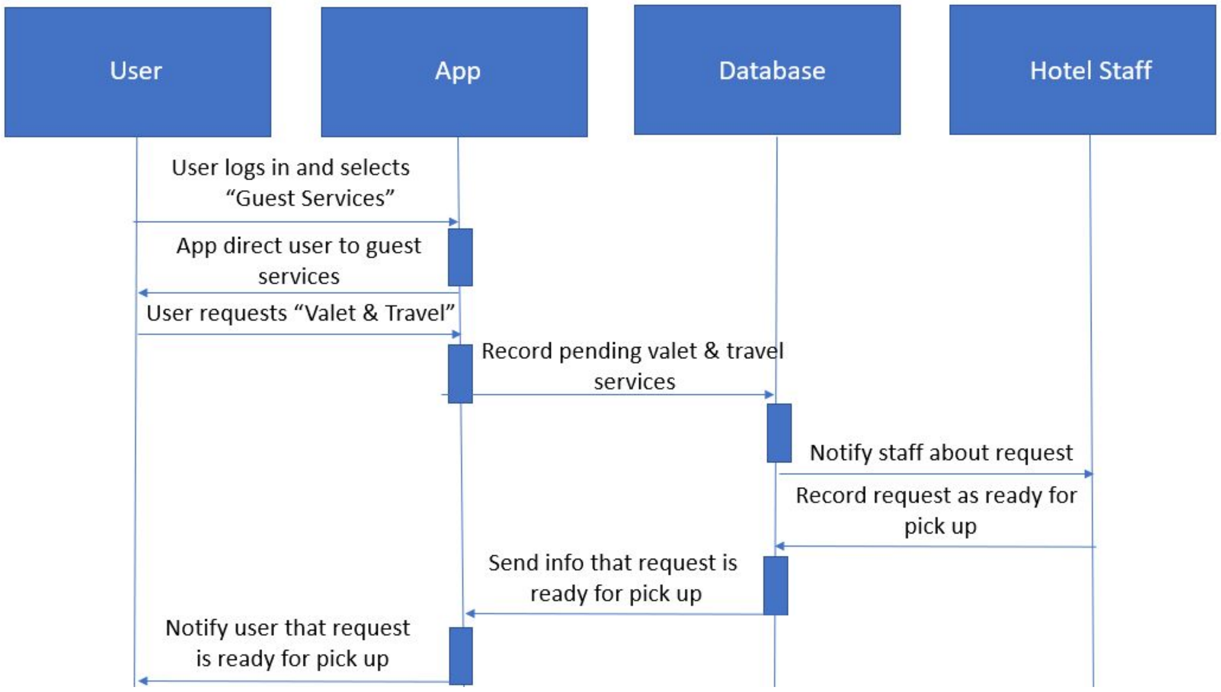
Use Case 9: Request Status


User
App
Database
User has logged in and makes a request from one of those offered in the app
System receives request and labels request as "in progress"
User reports to the system when service has been completed
System marks request as "granted"

Use Case 11: Maintenance

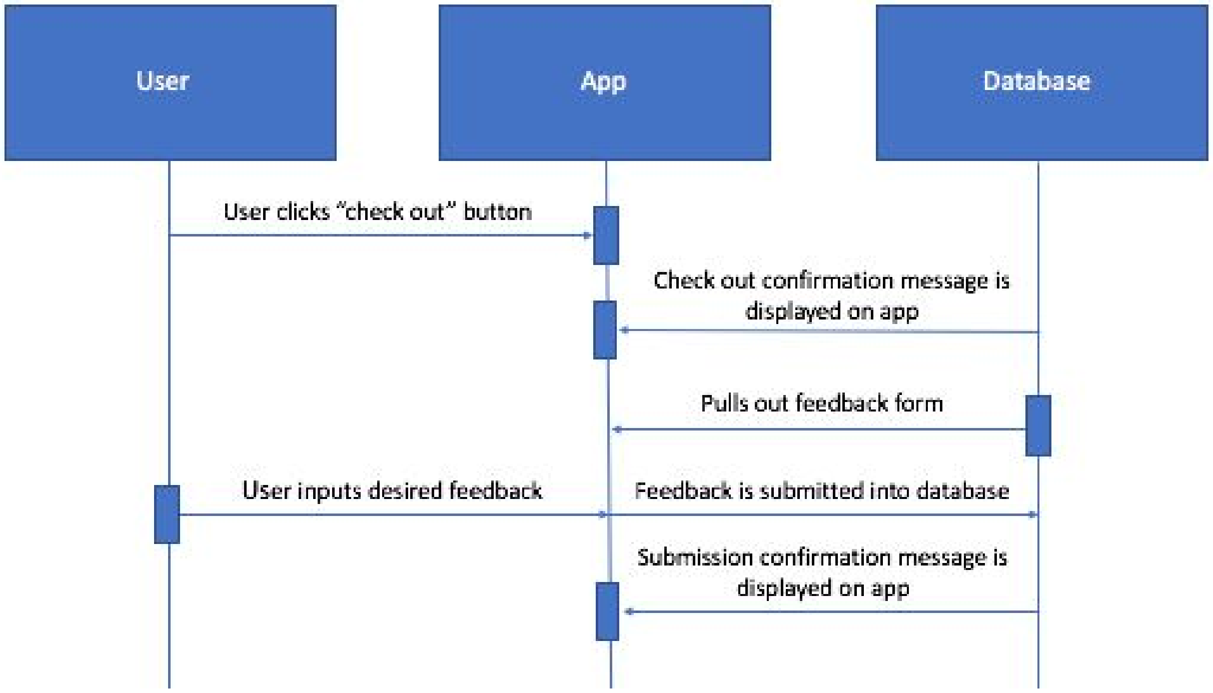

User
App
Database
User logs in and selects "Maintenance Requests"
Database pulls options for maintenance requests
App will submit option into database
User will be able to select from the options presented
Database will send confirmation message

User Case 12: Valet & Travel Services



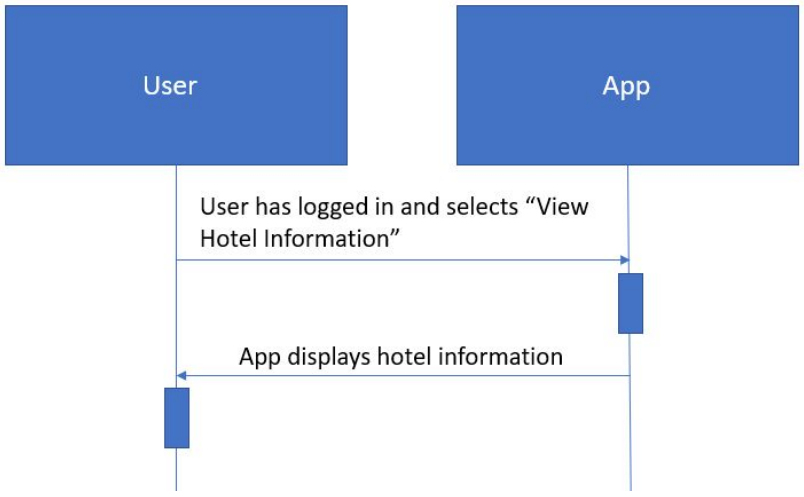
User
App
Database
Hotel Staff
User logs in and selects "Guest Services"
App direct user to guest services
User requests "Valet & Travel"
Record pending valet & travel services
Notify staff about request
Record request as ready for pick up
Send info that request is ready for pick up
Notify user that request is ready for pick up

User Case 13: Feedback

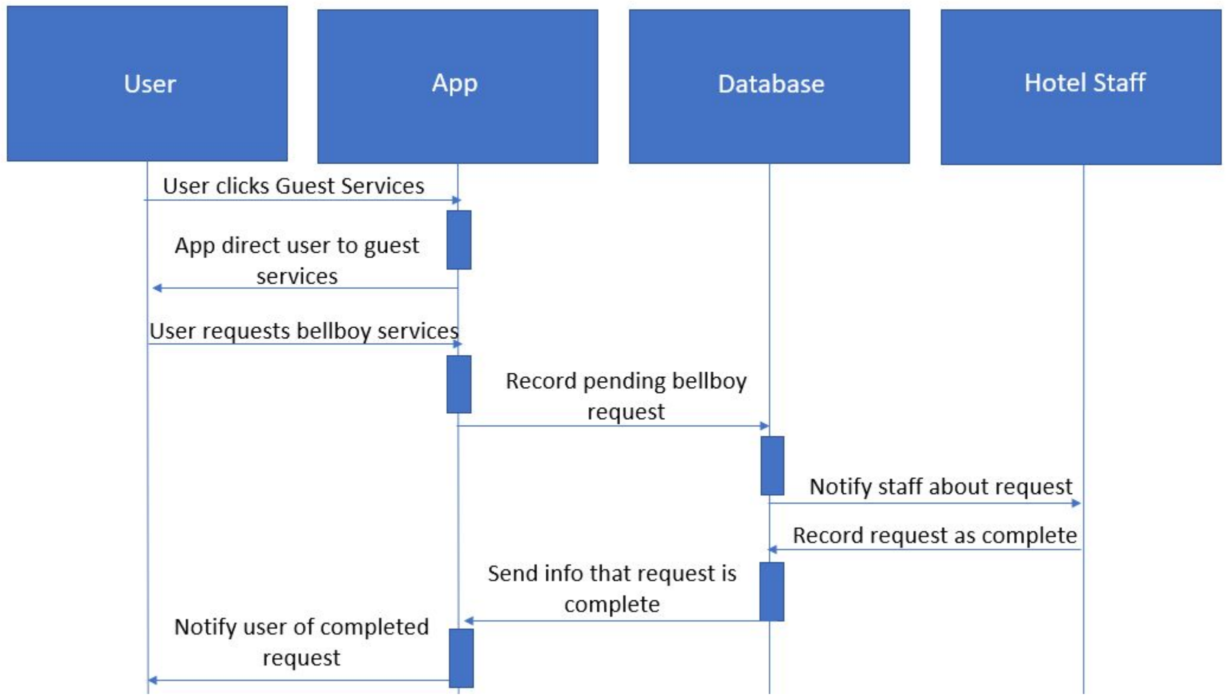


User
App
Database
User clicks "check out" button
Check out confirmation message is displayed on app
Pulls out feedback form
User inputs desired feedback
Feedback is submitted into database
Submission confirmation message is displayed on app

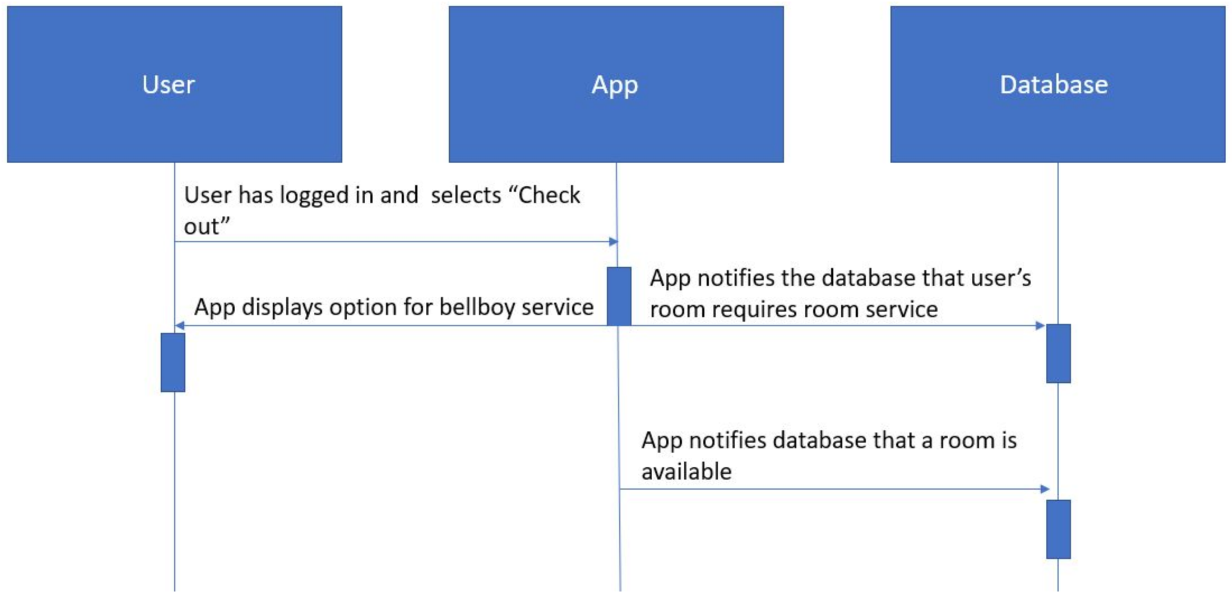
User Case 14: Hotel Information


User
App
User has logged in and selects "View Hotel Information"
App displays hotel information

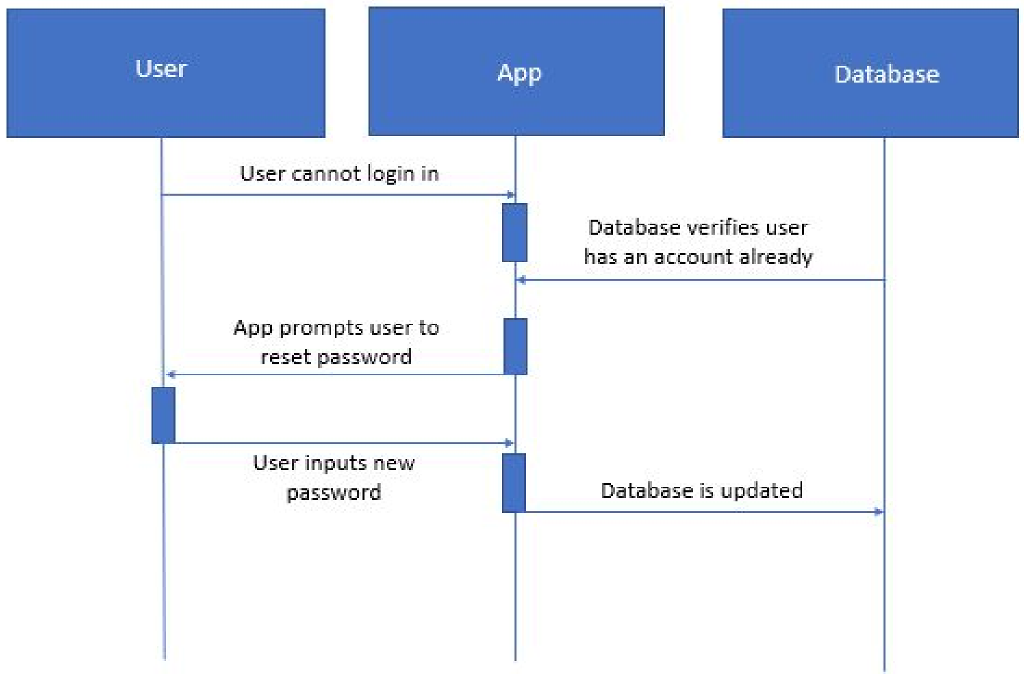
User Case 15: Bellboy Services


User
App
Database
Hotel Staff
User clicks Guest Services
App direct user to guest services
User requests bellboy services
Record pending bellboy request
Notify staff about request
Record request as complete
Send info that request is complete
Notify user of completed request

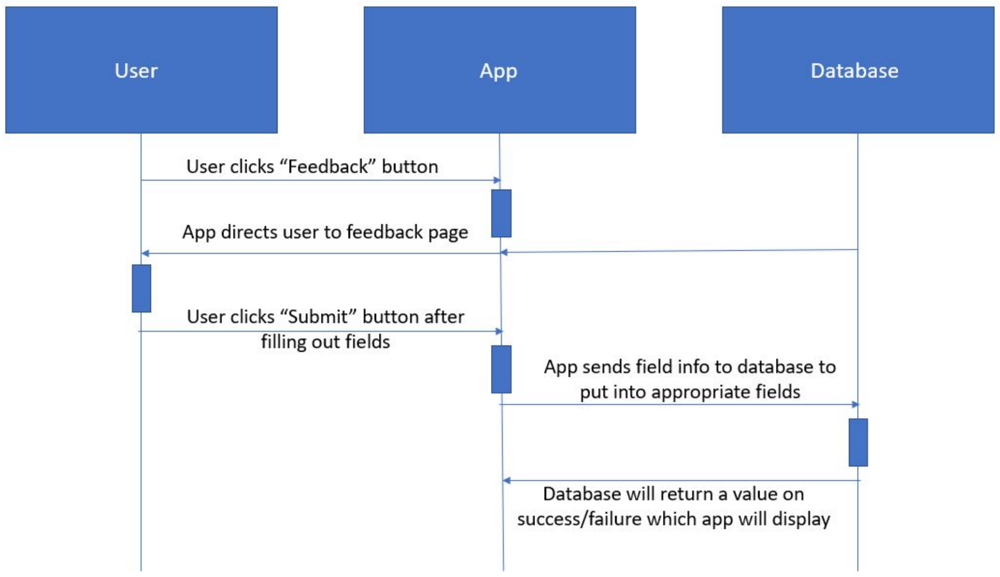
User Case 16: Checkout


User
App
Database
User has logged in and selects "Check out"
App notifies the database that user's room requires room service
App displays option for bellboy service
App notifies database that a room is available

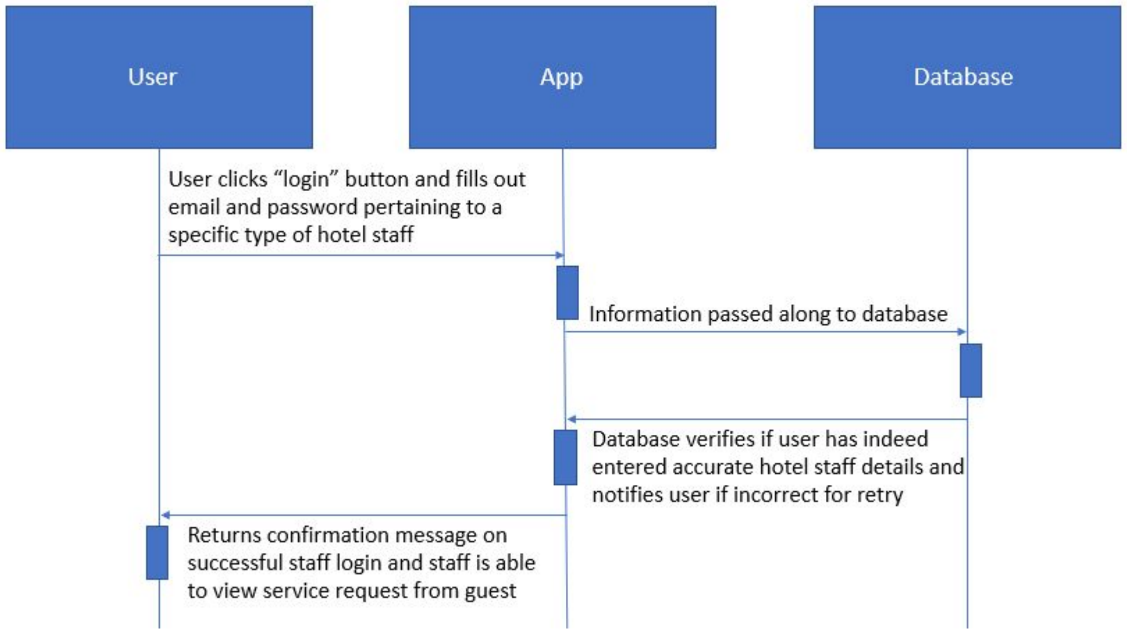
User Case 17: Forgot Password


User
App
Database
User cannot login in
Database verifies user has an account already
App prompts user to reset password
User inputs new password
Database is updated

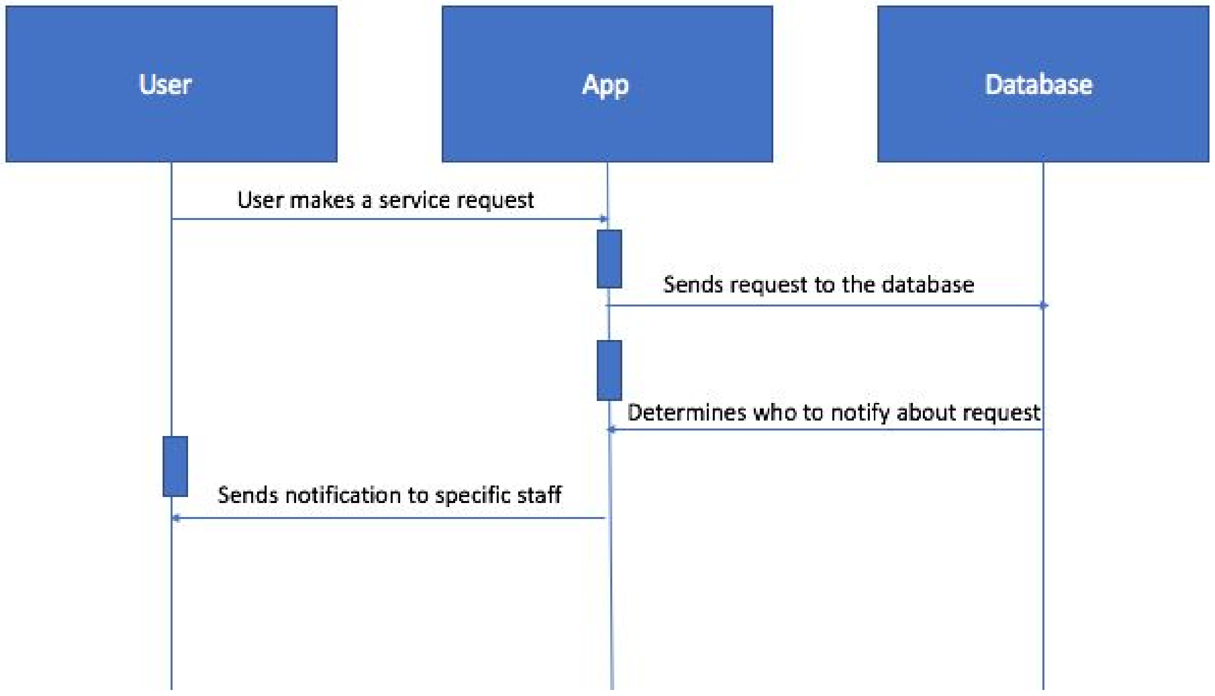
Use Case 18: Feedback


User
App
Database
User clicks "Feedback" button
App directs user to feedback page
User clicks "Submit" button after filling out fields
App sends field info to database to put into appropriate fields
Database will return a value on success/failure which app will display

Use Case 19: Preset Manager Accounts for Hotel Staff


User
App
Database
User clicks "login" button and fills out email and password pertaining to a specific type of hotel staff
Information passed along to database
Database verifies if user has indeed entered accurate hotel staff details and notifies user if incorrect for retry
Returns confirmation message on successful staff login and staff is able to view service request from guest

Use Case 20: Send Service Request to Appropriate Staff Account


User
App
Database
User makes a service request
Sends request to the database
Determines who to notify about request
Sends notification to specific staff

3.3.4 Traceability Matrix

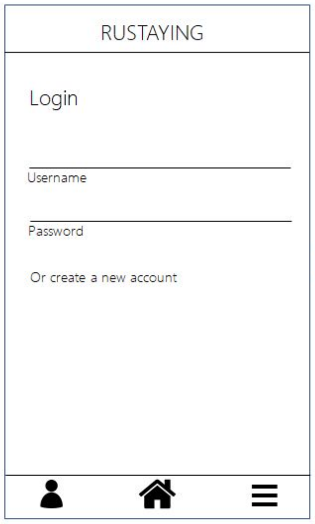
Reqs	P.W	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20
1	2	X	X	X	X																
2	2		X	X	X		X					X									
3	6			X		X	X														
4	6				X																
5	4						X			X											
6	8				X			X													
7	5									X		X									
8	3				X																
9	2									X						X					
10	7					X															
11	1														X						
12	3									X			X								
13	10							X	X	X											
14	1													X							
15	3																				
16	6																X				
17	1										X										
18	1																	X			
19	2							X													
20	1																		X		
21	1																			X	
22	1		X																		X
23	6					X															
24	1									X											
25	1															X					
26	2																				
27	1																				
28	1																				
29	1																				
30	1				X																
31	1																				
32	3																				
Max P.W.		2	2	6	8	7	6	10	10	10	1	5	3	1	1	2	6	1	1	1	1
Total P.W.		2	4	10	22	19	12	20	10	25	1	7	3	1	1	3	6	1	1	1	1

Chapter 4

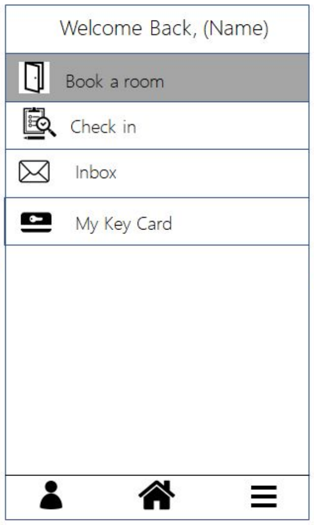
User Interface Specification

4.1 Preliminary Design

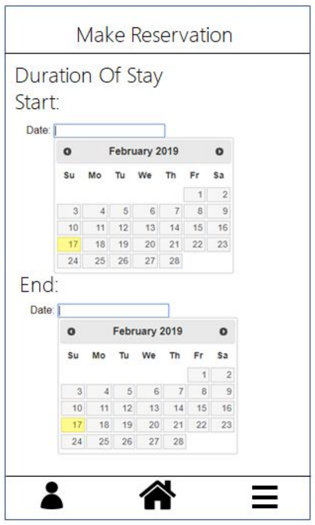
Use Case 4: Make Reservation



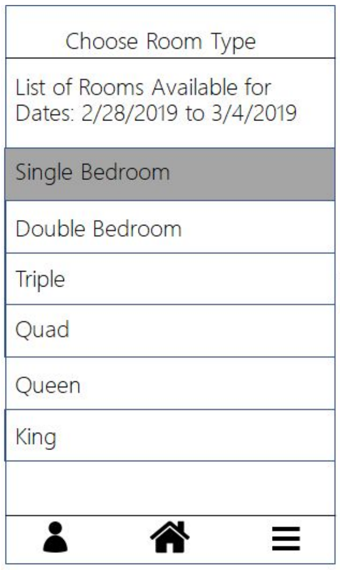
Start page where user can login



User selects Book a Room

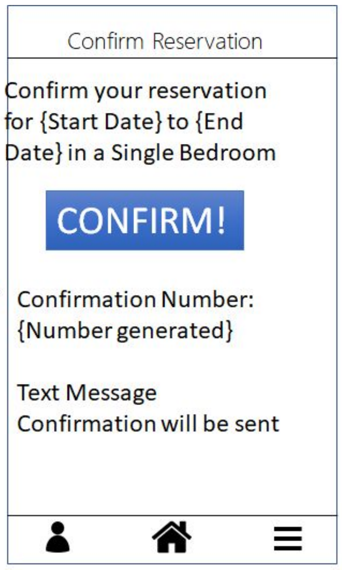


User selects start and end date



List of Rooms Available for Dates: 2/28/2019 to 3/4/2019

User selects from available rooms



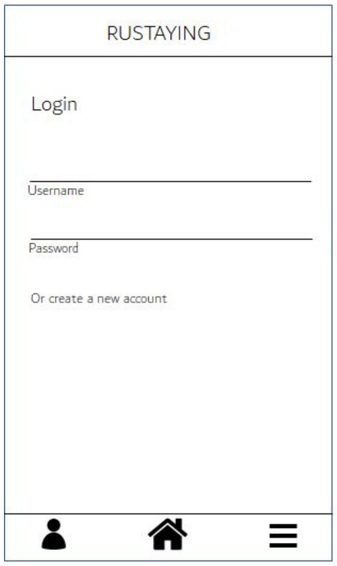
Confirm your reservation for {Start Date} to {End Date} in a Single Bedroom

Confirmation Number: {Number generated}

Text Message Confirmation will be sent

User confirms booking

Use Case 11: Maintenance Request


RUSTAYING

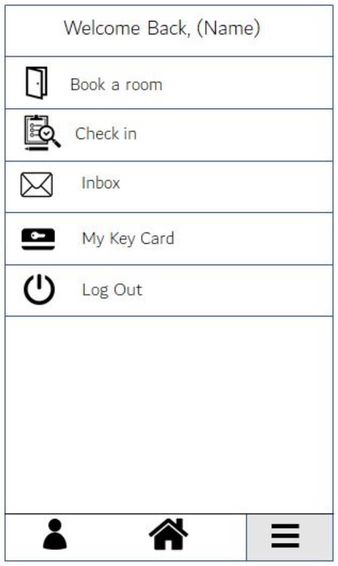
Login

Username

Password

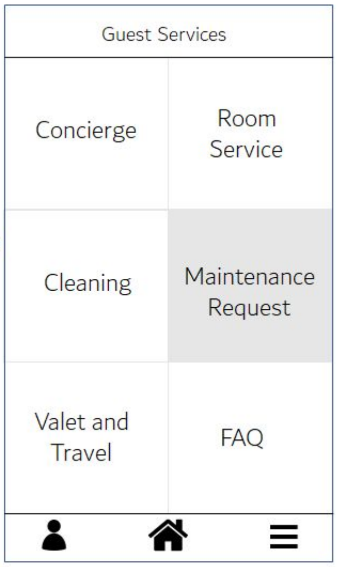
Or create a new account

User logs in


Welcome Back, (Name)

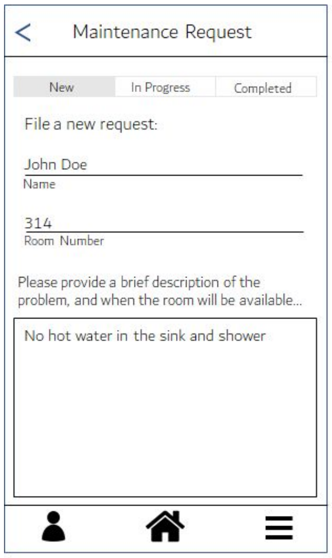
- Book a room
- Check in
- Inbox
- My Key Card
- Log Out

Select the Guest Services


Guest Services

Concierge	Room Service
Cleaning	Maintenance Request
Valet and Travel	FAQ

Select Maintenance Request


< Maintenance Request

New | In Progress | Completed

File a new request:

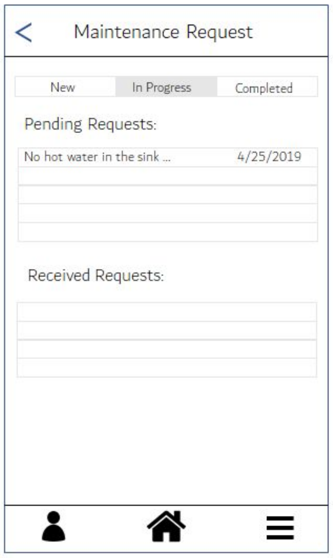
John Doe
Name

314
Room Number

Please provide a brief description of the problem, and when the room will be available...

No hot water in the sink and shower

New Requests


< Maintenance Request

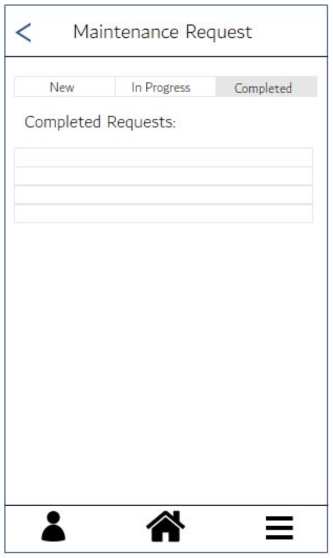
New | In Progress | Completed

Pending Requests:

No hot water in the sink ...	4/25/2019

Received Requests:

Pending/In progress Requests


< Maintenance Request

New | In Progress | Completed

Completed Requests:

Completed Requests

4.2 User Effort Estimation

Use Case Weights

Category	**Description**	**Weight**
Simple	Simple user interface with up to **one** participating actor and initiating actor. Success scenario has **<=3 steps** Domain model includes **<=3 concepts**	5
Average	Moderate user interface design with **two or more** participating actors. Success scenario has **4-7 steps** Domain model includes **5-10 concepts**	10
Complex	Complex user interface or processing with **three or more** participating actors. Success scenario has **>=7 steps** Domain model includes **>=10 concepts**	15

Use Case	**Description**	**Category**	**Weight**
Register (UC1)	Average user interface. 5 steps for the main success scenario. 2 participating actors (Application User and Database)	Average	10
Login (UC2)	Simple user interface. 3 steps for the main success scenario. 2 participating actors (Application User and Database)	Simple	5
Logout (UC3)	Simple user interface. 3 steps for all scenarios. 2 participating actors (Application User and Database)	Simple	5
Make a Reservation (UC4)	Average user interface. 5 steps for the main success scenario. 2 participating actors (Application User and Database)	Average	10
Check In (UC5)	Simple user interface. 2 steps for the main success scenario. 3 participating actors (Application User, Database, and Hotel Staff)	Simple	5
Room	Average user interface. 4 steps for the main success	Average	10

Service (UC6)	scenario. 3 participating actors (Application User, Database, and Hotel Staff)		
Check Availability of Rooms (UC7)	Simple user interface. 3 steps for the all scenarios. 2 participating actors (Application User and Admin)	Simple	5
Review Guest Data (UC8)	Average user interface. 4 steps for the all scenarios. 2 participating actors (Admin and Database)	Average	10
Request Status (UC9)	Average user interface. 4 steps for the all scenarios. 3 participating actors (Application User, Database and Hotel Staff)	Average	10
Maintenance (UC11)	Average user interface. 5 steps for the main success scenario. 3 participating actors (Application User, Database, and Hotel Staff)	Average	10
Valet & Travel (UC12)	Average user interface. 4 steps for the main success scenario. 3 participating actors (Application User, Hotel Staff, and Car)	Average	10
Feedback (UC13)	Average user interface. 5 steps for the main success scenario. 2 participating actors (Application User and Database)	Average	10
Hotel Information (UC14)	Simple user interface. 2 steps for the main success scenario. 2 participating actors (Application User and Database)	Simple	5
Bellboy Services (UC15)	Average user interface. 4 steps for the main success scenario. 2 participating actors (Application User and Hotel Staff)	Average	10
Checkout (UC16)	Average user interface. 4 steps for the main success scenario. 3 participating actors (Application User, Database, and Hotel Staff)	Average	10
Forgot Password (UC17)	Simple user interface. 3 steps for main success scenario. 2 participating actors (Application User and Database)	Simple	5
Report Feedback	Average user interface. 1 step for main success scenario. 2 participating actors (Manager and Database)	Average	10

Data (UC18)			
Preset Manager Accounts (UC19)	Simple user interface. 2 steps for main success scenario. 2 participating actors (Manager and Database)	Simple	5
Service Request sent to manager account (UC20)	Simple user interface. 2 steps for main success scenario. 2 participating actors (Application User and Database)	Simple	5

UUCP = 9 x Simple + 11 x Average = 6 x 5 + 10 x 10 = **<u>155 UUCP</u>**

Technical Complexity Factors

Technical Factor	**Description**	**Weight**	**Perceived Complexity**	**Calculated Factor**
T1	Adaptable to different types of Android phones	2	3	6
T2	Performance Objectives	1	2	2
T3	Reusable design of UI	1	2	2
T4	Reusable design of code (Object oriented)	3	3	9
T5	Concurrent use by multiple users	2	1	2
T6	Internal data management	3	2	6
T7	Ease of use by user	1	1	1
T8	End User Efficiency	1	1	1
			Technical Factor Total:	29

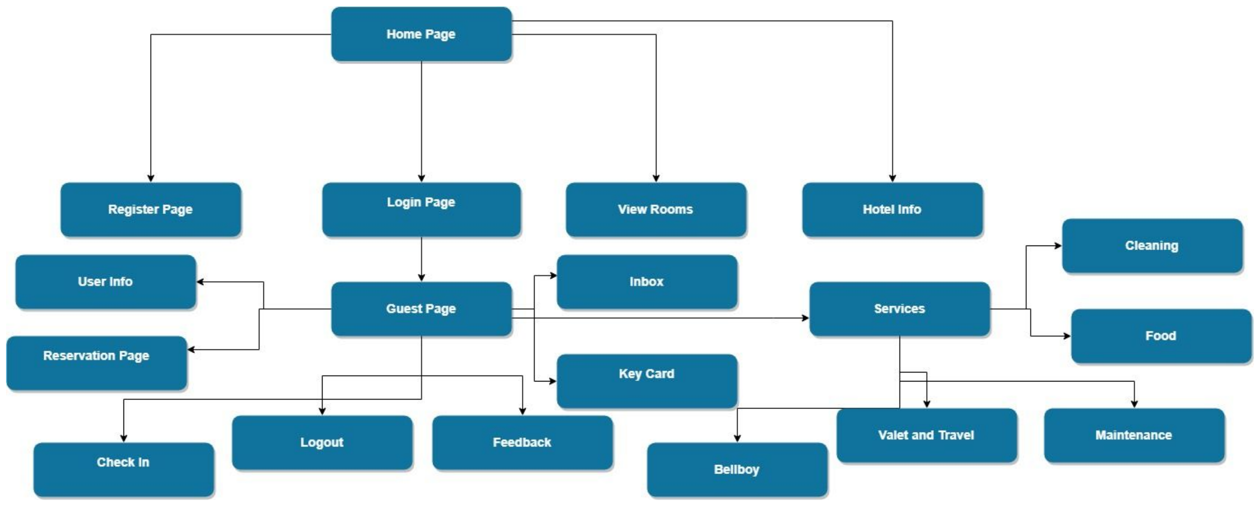
TCF = 0.6 + (0.01 * Technical Factor Total)
TCF = 0.6 + (0.01 * 29)
TCF = 0.89

Total Use Case Points

UUCP = 155
TCF = 0.89
ECF = 1
UCP = UUCP x TCF x ECF
UCP = 155 x 0.89 x 1 = 115.7 = **137 Use Case Points**

The final use points calculated was 137. This is an estimation of the size of our project based on our use cases listed above and the relative complexity of each one. In addition to the functionality, we also considered the technical complexities of the non-functional aspects. Based on the technical factor, the number of people in our group and our specific skill set, we estimated a proper perceived complexity for each factor.

4.2.1 Navigation Tree



Since we are building an android app navigation through each page is one click that will take you to a new page in the app. So from the guest services page to access each of the different features it is one button click. Each page also provides a back button to return to the previous one.

Chapter 5

Domain Analysis

5.1 Domain Model

The domain model is a conceptual model of a domain that incorporates both behavior and data. This model can be used to solve problems related to that domain. By using the requirements listed from each use case, the domain model shows exactly how the user will interact with the system and how the system will complete the requirement. It is helpful to map out the interactions from the user and also the interactions within the system itself.

5.1.1 Concept Definitions and Responsibilities

Responsibility Description	**Type**	**Concept Name**

RS1. Main source for all other subsystems to interact with. Coordinate actions and delegate work based on user interactions.	D	Controller
RS2. Verify if user credentials are valid	D	UserVerification
RS3. Create a new guest account	K	UserManagement
RS5. Store user account information (username, password, email, previous reservations etc…)	D	UserManagement
RS4. To allow user to make a reservation	D	RoomControl
RS6. To allow the guest to check-in for their hotel reservation	K	RoomControl
RS7. Guest can request room service	D	HotelServices
RS8. Guest can make maintenance requests for their room during their stay	D	HotelServices
RS9. Guest can request a bellboy for luggage	K	HotelServices
RS10. Guest can use the hotel car service	K	HotelServices
RS11. System will keep track of available / unavailable rooms	D	RoomTracking
RS12. System will keep track of each hotel service as it is requested	D	HotelTracking
RS13. Review all data on services used by guests	D	AdminControl
RS14. Present data in viewable manner to analysis and predictions	D	AdminControl

Some concepts interact directly with the database but serve different functions so they are listed as separate concepts. The concepts that directly interact with the database are: UserManagement, RoomTracking, HotelTracking and AdminControl. The other concepts are control concepts that the user can interact with through the controller to achieve functionality.

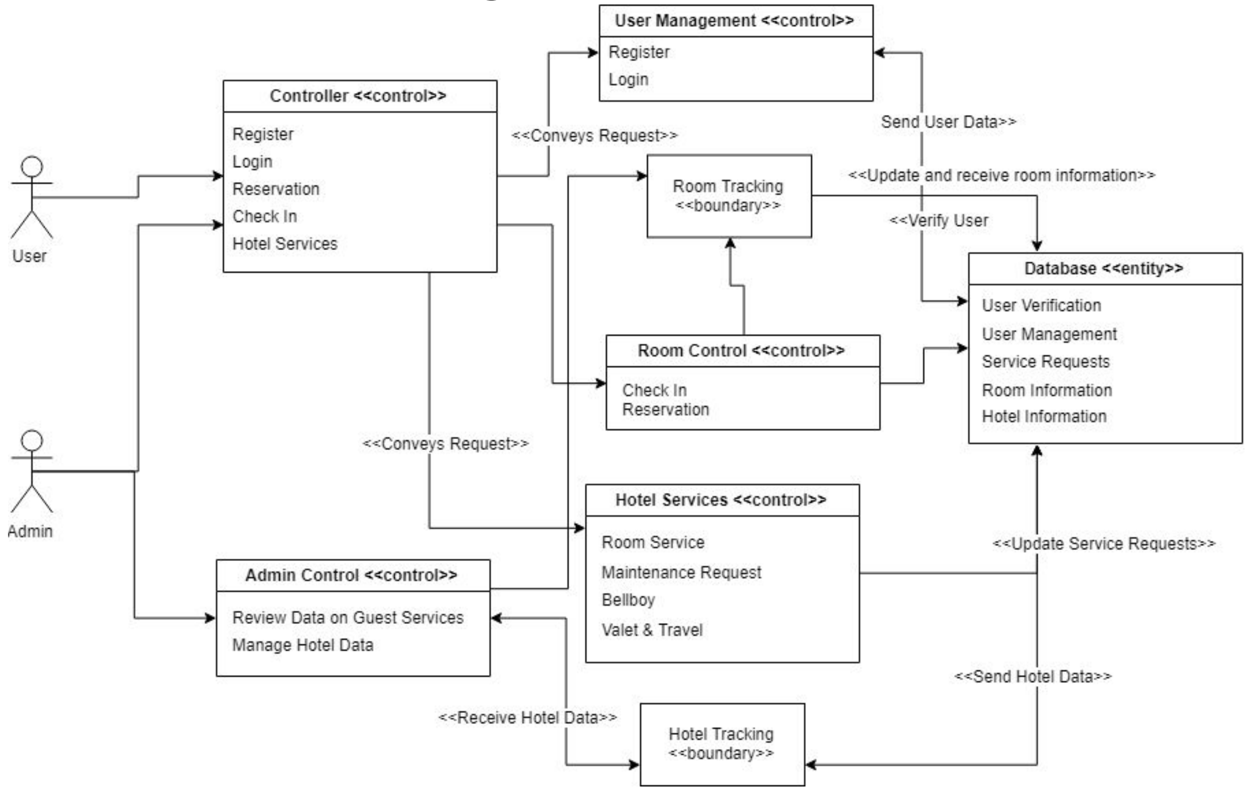
5.2 Concept Associations

Concept Pair	Association Description	Association Name
UserVerification ↔ UserManagement	User credentials sent to UserManagement to check if it is valid and receives back a confirmation	Validate User
Controller ↔ RoomControl	As user interacts with app, the controller will send room reservations and room check-in information to RoomControl concept	Room Bookings
RoomControl ↔ RoomTracking	As RoomControl is used, the information is passed to RoomTracking, which will store the data and also allow for analysis	Room Analysis
Controller ↔ HotelServices	When users request any sort of service available through our app, the task will be delegated to HotelServices concept, which will appropriately handle the request	Handle Service Request
HotelServices ↔ HotelTracking	As HotelServices handles the guest requests, the data will be sent to HotelTracking, which will store the data and allow for analysis	Hotel Analysis
RoomTracking ↔ AdminControl	Once room data is reported in RoomTracking concept, it will be sent to AdminControl, where the admin can easily view it	Admin Room Data
HotelTracking ↔ AdminControl	Once hotel services data is reported in HotelTracking, it is sent over to AdminControl and presented so that the admin can easily notice trends and patterns	Admin Hotel Data

5.3 Attribute Definitions

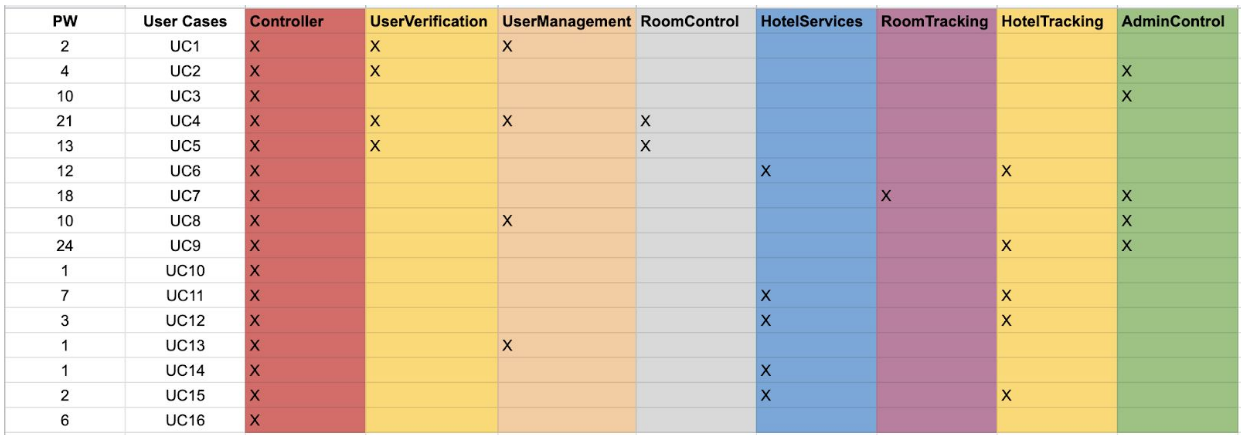
Concept	Attributes	Attribute Definitions
UserVerification	User's account data	Contains user's identification including name, email, phone number, and home address associated with the account
UserManagement	Store account data	Stores newly made account or recently edited account information
RoomControl	Room data	Shows available rooms that are reservable
	Check In	Allows user to check in
HotelServices	Request data	Directs user requests to corresponding service
RoomTracking	All room data	Contains data on available and unavailable rooms
HotelTracking	All Request data	Contains data from all user requests in the hotel
AdminControl	Hotel data	Contains information on all in progress or completed hotel services

5. 4 Domain Model Diagram



The domain model was derived by looking at our concept definitions and which use cases they fulfilled. The main component in our domain model is the controller that connects to all other concepts in our device so it fulfills all our use cases. From the controller the user can access the other concepts in our application. The main ones being user management, room control, and hotel services. From user management the user can register and log in to the app to access the different components of our app. The next one room control fulfills UC-4 and UC-5. The hotel services controller gives users access to the various features described in our use cases that a user can do. The database serves as the entity that stores all this information about user, hotel information, room information, requests, and fulfills requests to send and store data. Finally the admin control allows the admin of the hotel to have additional features to fulfill our use cases. With boundaries such as room tracking and hotel tracking that communicate with the database the admin control allows the admin to have access to all hotel data sent in by guests and retrieve it to perform future predictions about management and efficiency.

5.5 Traceability Matrix for Domain Concepts



PW	User Cases	Controller	UserVerification	UserManagement	RoomControl	HotelServices	RoomTracking	HotelTracking	AdminControl
2	UC1	X	X	X					
4	UC2	X	X						X
10	UC3	X							X
21	UC4	X	X	X	X				
13	UC5	X	X		X				
12	UC6	X				X		X	
18	UC7	X					X		X
10	UC8	X		X					X
24	UC9	X						X	X
1	UC10	X							
7	UC11	X				X		X	
3	UC12	X				X		X	
1	UC13	X		X					
1	UC14	X				X			
2	UC15	X				X		X	
6	UC16	X							

5.6 System Operation Contracts

UC-1 - Register an account

- Preconditions:
  - Email guest is using to register has not been previously used
  - Password meets security standards
- Postconditions:
  - User now has an account that can be used to login into the application

UC-2 - Login to account

- Preconditions:
  - User has registered an account
- Postconditions:
  - User can enter email and password to login

UC-3 - Logout

- Preconditions:
  - User has registered an account
  - User is currently logged into the application
- Postconditions:
  - A different user can now log in or register an account

UC-4 - Make a reservation

- Preconditions:
  - Guest must have an account be logged into the application
- Postconditions:
  - User will receive room options
  - Guest will receive an email confirmation of their reservation

UC-5 - Check-in

- Preconditions:
  - Guest has made a reservation
  - Guest has arrived at the hotel
- Postconditions:
  - Guest can access service of the hotel they are currently staying at

UC-6 - Room Service

- Preconditions:
  - Guest has checked into their hotel
- Postconditions:
  - Guest will have their request granted
  - Guest will be notified once their Room service request has been complete

UC-7 - Check Availabilities of Rooms

- Preconditions:
    - User must be logged into the application
- Postconditions:
    - Application will display what rooms are occupied and what rooms are not

UC-8 - Review Guest/Service Data

- Preconditions:
    - Have an Admin account
    - Have Guests staying at your hotel
- Postconditions:
    - Display Guest Data (ex: number of Guests, their rooms, etc.)
    - List of Requested Services per Guest
    - List of Rooms and their occupants

UC-9 - Request Status

- Preconditions:
    - Guest must have made a request for some type of service
- Postconditions:
    - Request will be labeled "in-progress", "granted", or "denied"

UC-11 - Maintenance

- Preconditions:
    - Guest is logged in and is on the tab to submit maintenance requests
- Postconditions:
    - Guest will have maintenance request fullfilled
    - Guest will receive confirmation message once maintenance request is fulfilled

UC-12 - Valet & Travel Services

- Preconditions:
    - Guest is logged into the application
- Postconditions:
    - Guest will have their car, driven out or into the garage
    - Guest can arrange transportation and will be notified once it is ready

UC-13 - Feedback

- Preconditions:
    - Guest has checked out of hotel
- Postconditions:
    - Guest will be prompted with a feedback form

UC-14 - Hotel Information

- Preconditions:
    - Guest is logged in

- Postconditions:
    - Guest will be able to view a hotel’s information. (e.g. Restaurants, Facilities, Hours)

UC-15 - Bellboy Services

- Preconditions:
    - Guest is checked in
- Postconditions:
    - Guest will have luggage taken to their room
    - Guest will be notified once luggage has been taken up

UC-16 - Checkout

- Preconditions:
    - Guest has finished their stay at the hotel
- Postconditions:
    - Guest will leave hotel
    - Guest will be sent feedback form

UC-17 - Forgot Password

- Preconditions:
    - Guest already has an account
- Postconditions:
    - Guest is able to login with their new password

UC-18 - Checkout

- Preconditions:
    - Admin has access to all feedback data
- Postconditions:
    - Admin is able to see a visual representation of the feedback data

UC-19 - Preset Manager accounts for hotel staff

- Preconditions:
    - Hotel staff or manager opens account
- Postconditions:
    - Each type of hotel service has its own admin account

UC-20 - Send service requests to appropriate manager account

- Preconditions:
    - Service requests made my guests
- Postconditions:
    - The service requests gets sent to the correct hotel staff account (ex: Bellboy request only shows up on bellboy account)

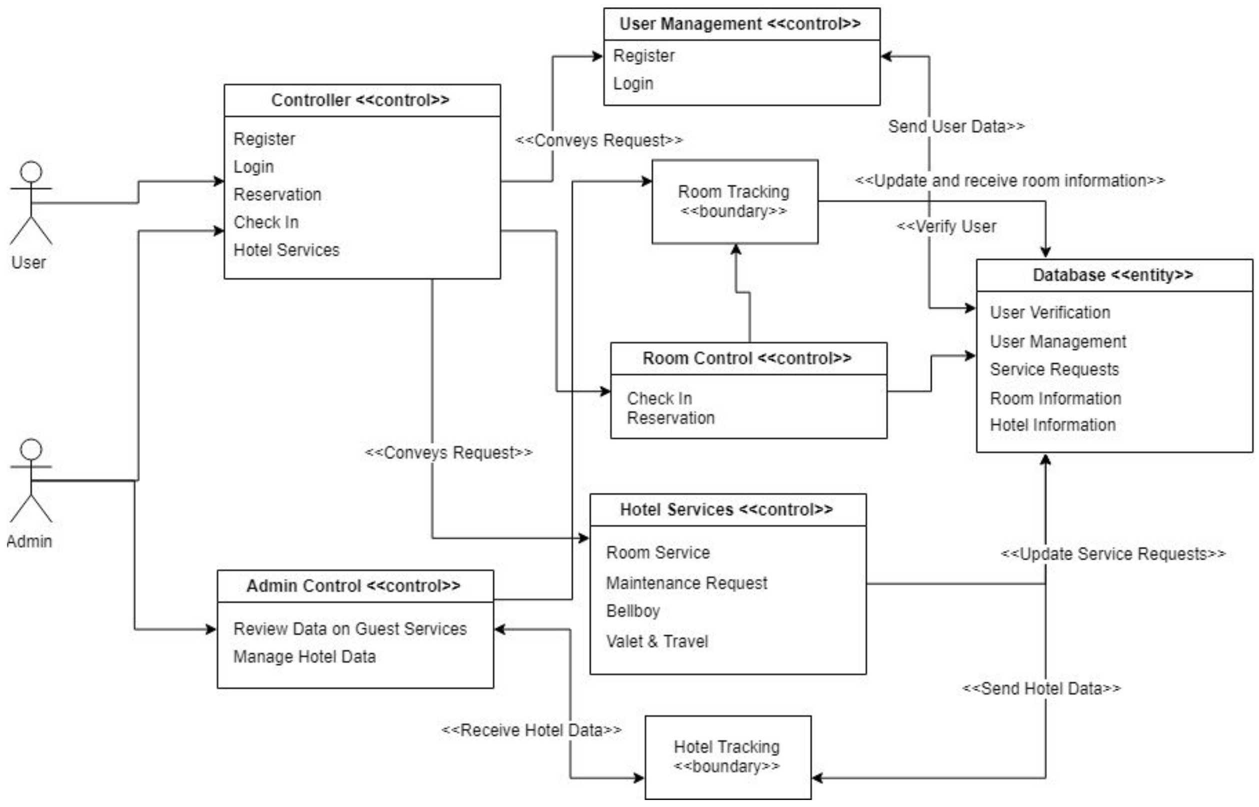
Chapter 6

Interaction Diagrams

6.1 Concept Definition and Responsibilities

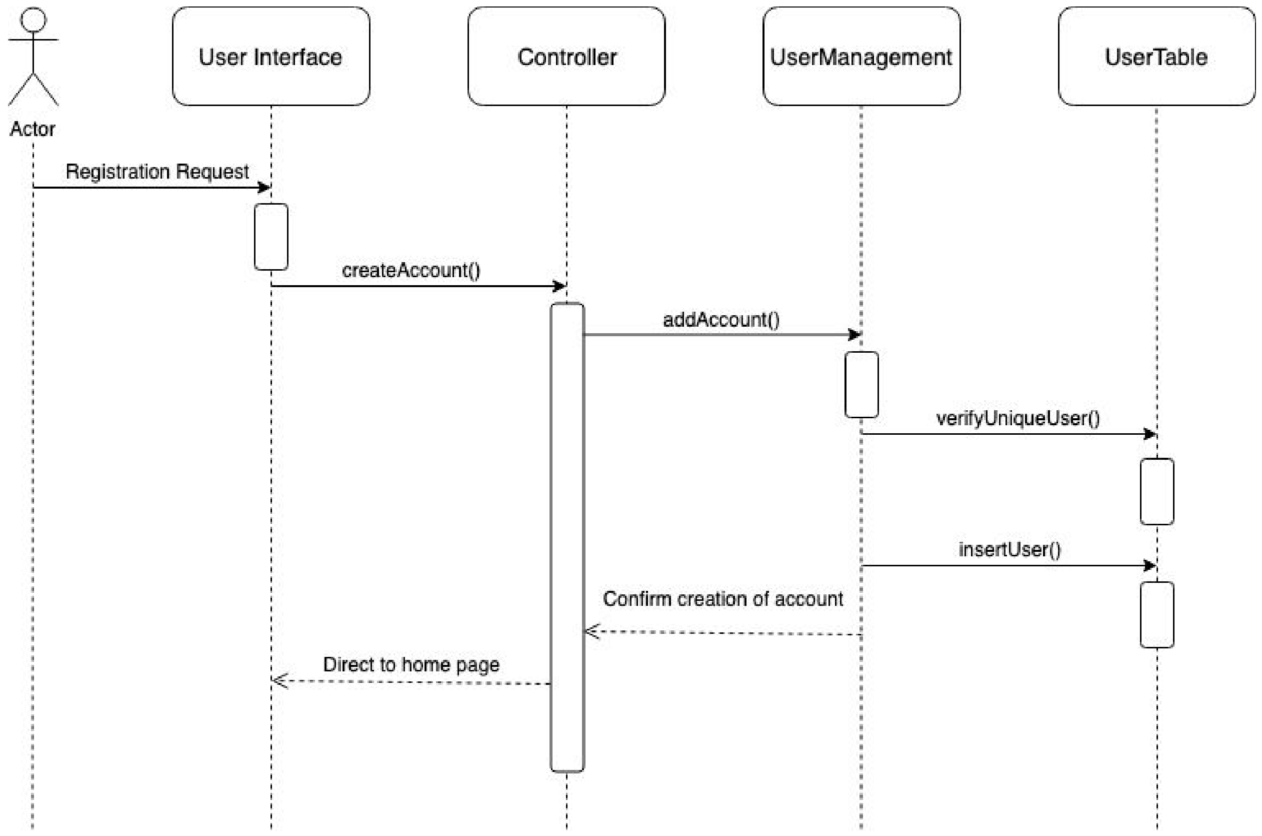
Responsibility Description	**Type**	**Concept Name**
RS1. Main source for all other subsystems to interact with. Coordinate actions and delegate work based on user interactions.	D	Controller
RS2. Verify if user credentials are valid	D	UserVerification
RS3. Create a new guest account	K	UserManagement
RS5. Store user account information (username, password, email, previous reservations etc…)	D	UserManagement
RS4. To allow user to make a reservation	D	RoomControl
RS6. To allow the guest to check-in for their hotel reservation	K	RoomControl
RS7. Guest can request room service	D	HotelServices
RS8. Guest can make maintenance requests for their room during their stay	D	HotelServices
RS9. Guest can request a bellboy for luggage	K	HotelServices
RS10. Guest can use the hotel car service	K	HotelServices
RS11. System will keep track of available / unavailable rooms	D	RoomTracking
RS12. System will keep track of each hotel service as it is requested	D	HotelTracking
RS13. Review all data on services used by guests	D	AdminControl
RS14. Present data in viewable manner to analysis and predictions	D	AdminControl

6.2 Domain Model Diagram



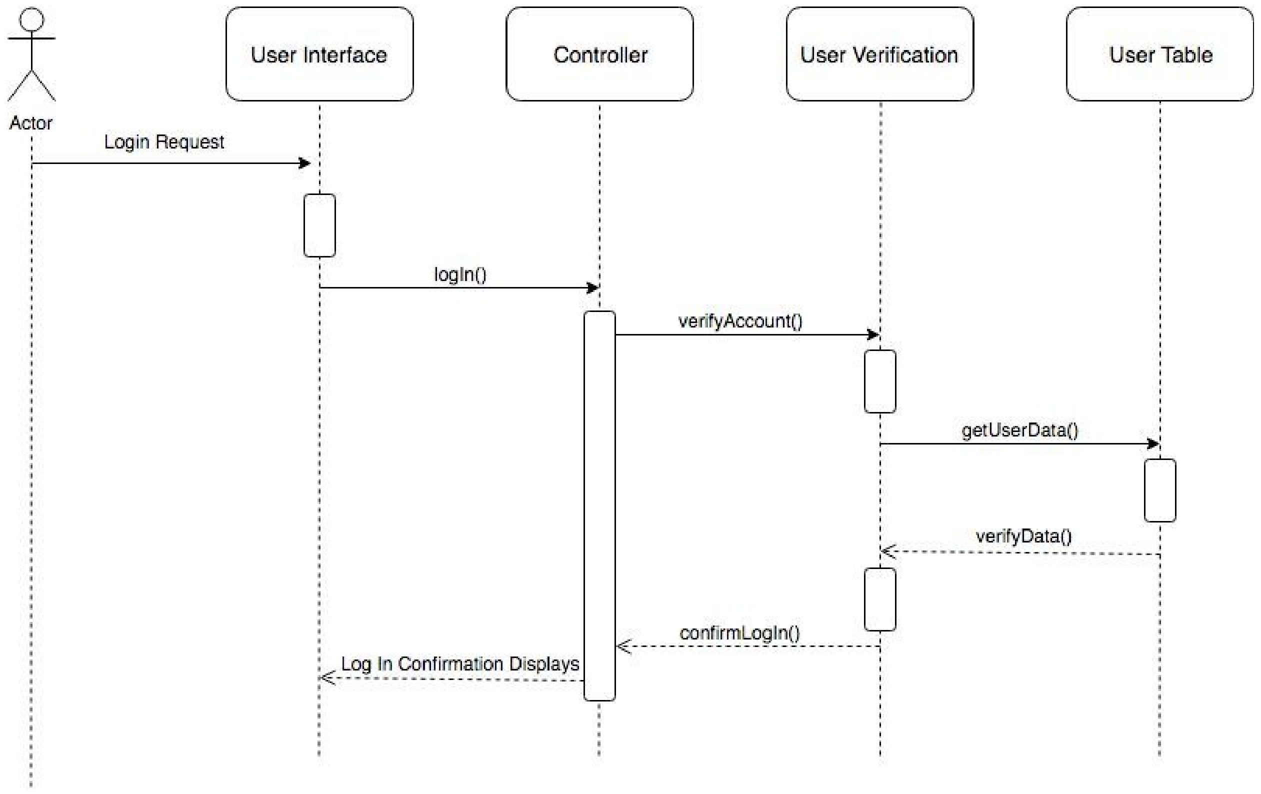
6.3 System Sequence Diagrams

Use Case 1 - Registration



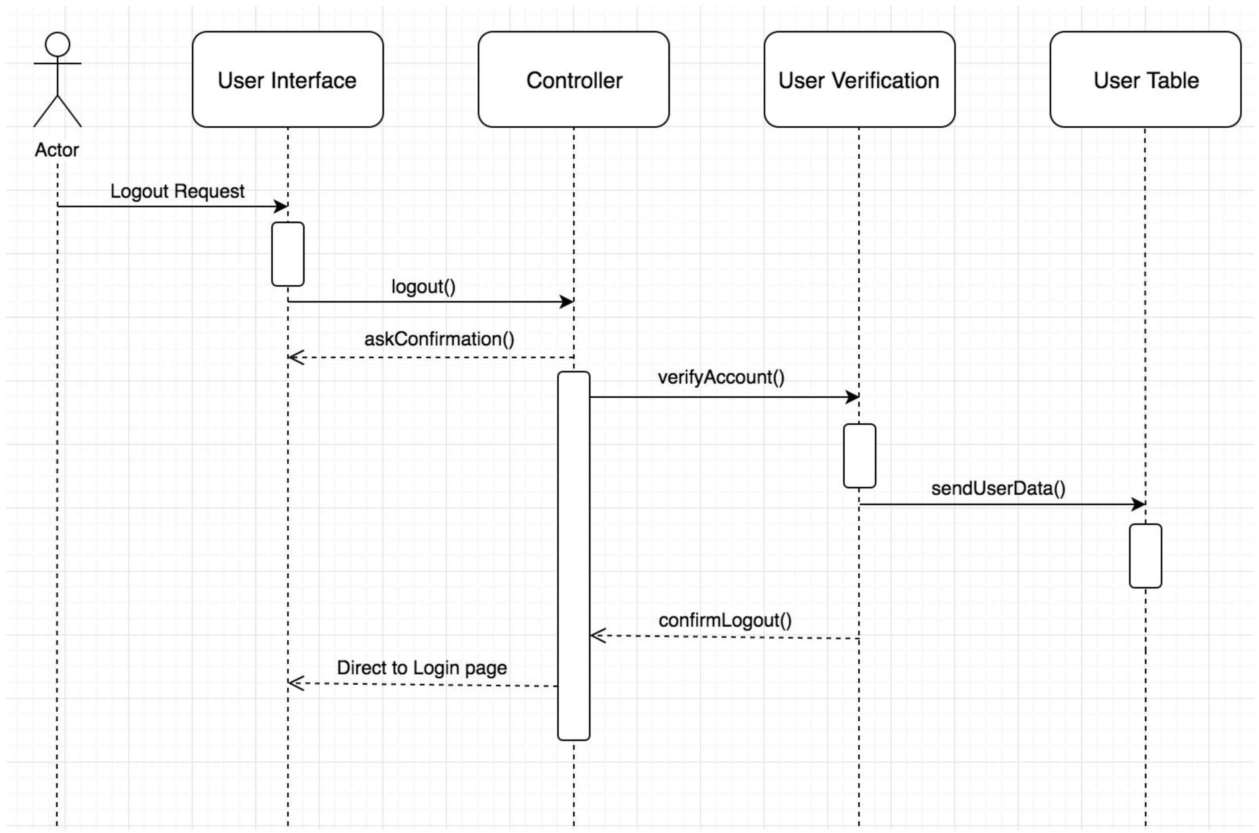
In this use case, the user can register for a new account. This is a relatively straight forward process. Once the user fills out the required information for a new account, the main controller will send the information to the UserManagement controller. The UserManagement controller is responsible for keeping track of all current and new accounts with the app RUStaying. It directly interacts with the database, specifically the UserTable, which is a table with all current accounts and their corresponding information. The UserTable concept was not listed in the domain model because we grouped all the specific database related entities. However, in these interaction diagrams, it is important to separate the different tables and information the database will be storing. The UserManagement controller will verify with the UserTable if this is a unique account and then process to insert the new user to the table. Once the main controller gets a confirmation that the user was created, it redirects the User Interface to the main home page so the user can access the hotel services through the app.

Use Case 2 - Login



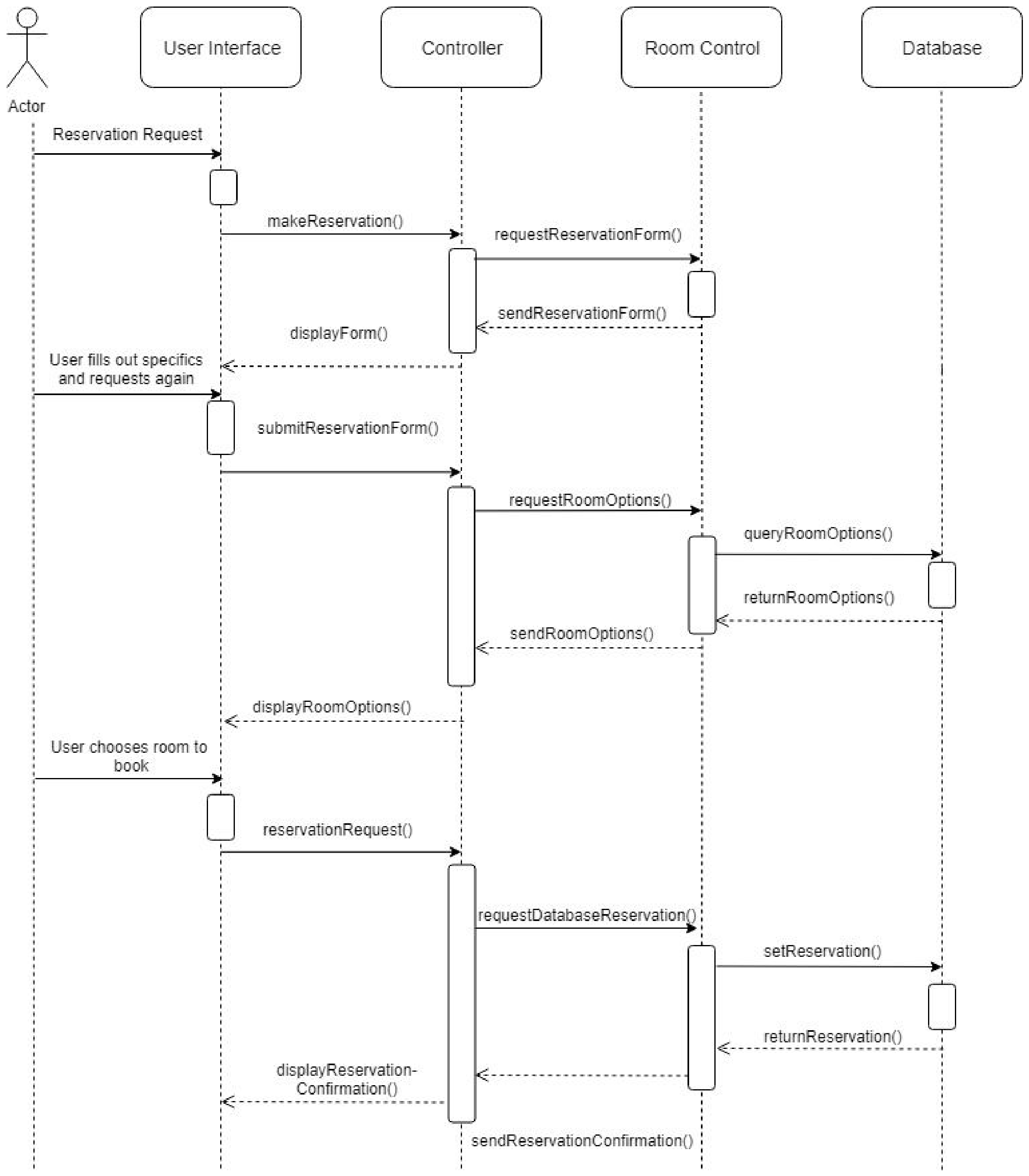
In this use case, the user can log into their account. Once the user inputs their username and password into the interface, the main controller will send the information into the UserVerification control. This is responsible for sending the data submitted by the user to the UserTable and verify if the information entered corresponds with an account that is in the database. If the account information is confirmed, the UserTable will send a verification to the UserVerification if the account information matches that of an existing account. The UserVerification will then confirm the login and allow the user to log in. If the information entered by the user does not correspond, then the user will be asked to input the login information again.

Use Case 3 - Logout



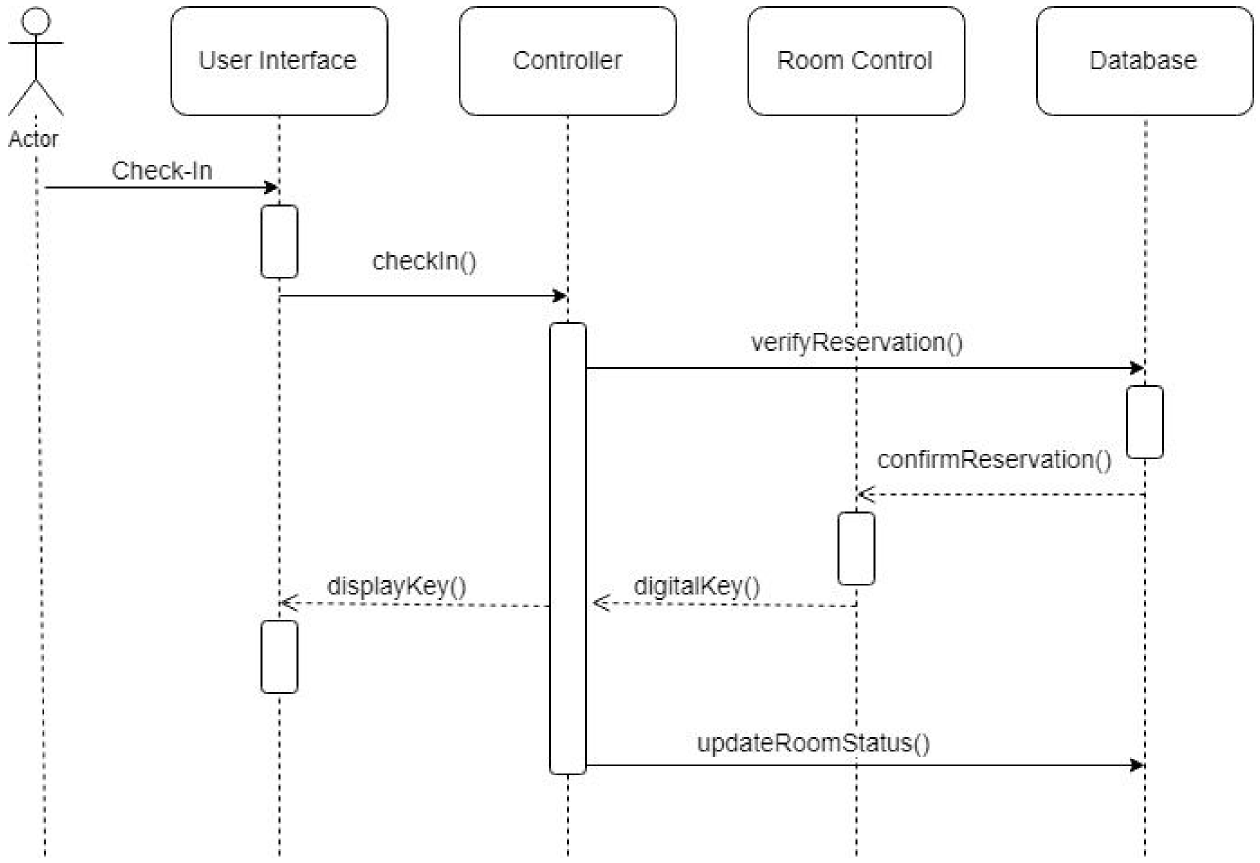
In this user case, the user is able to log out of an existing account. Once the user creates a logout request, the main controller will send a confirmation message back to the user to confirm logout. Then, the controller will send the information UserVerification control. User Verification will send the data submitted by the user to the User Table. The UserVerification will then confirm the logout and allow the user to logout of their account. The user will be redirected to the login page.

Use Case 4 - Make a Reservation



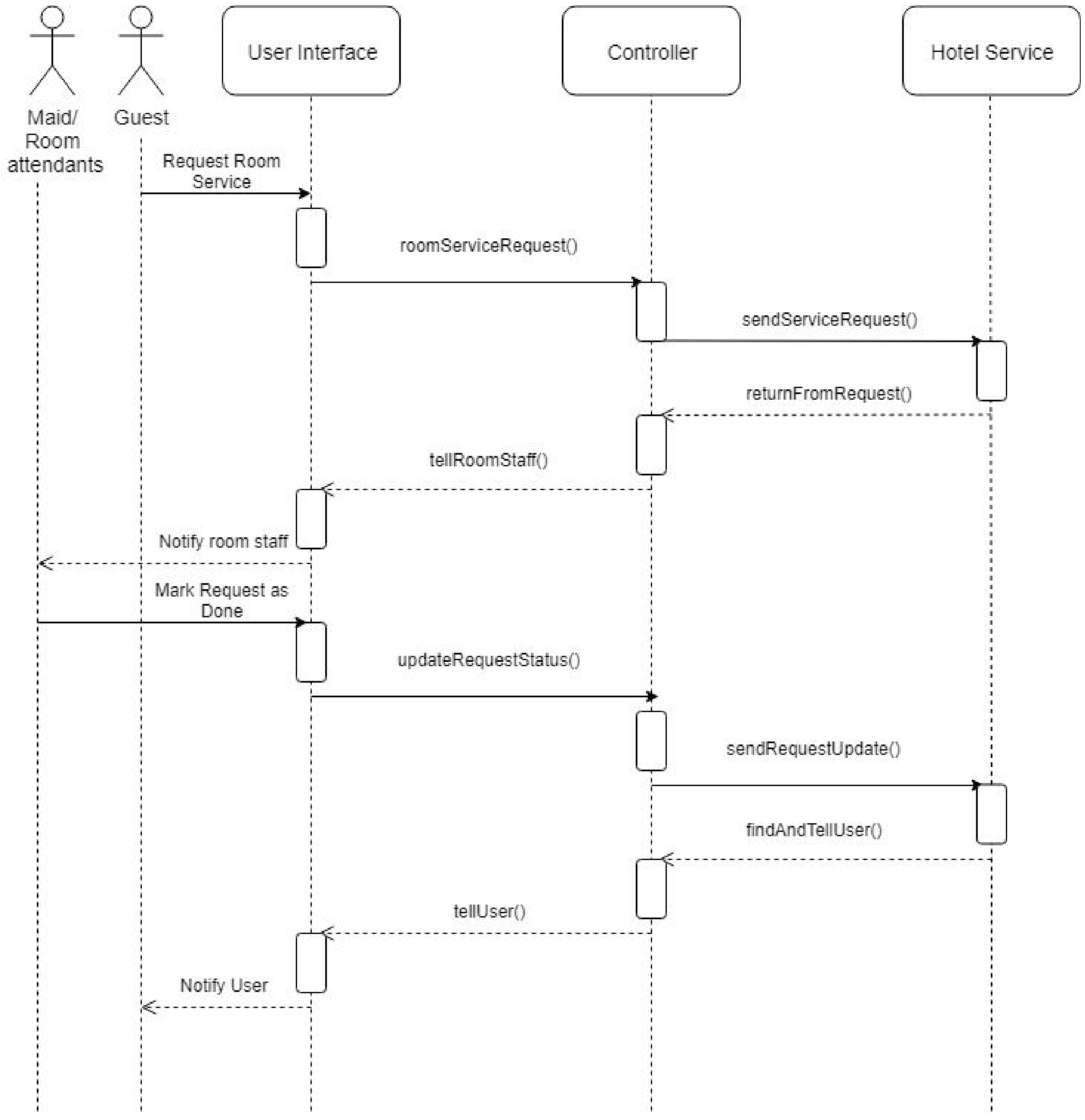
In this use case, the user will be able to make/book a reservation. This process can be roughly broken down into 3 parts. The first part, user will request to make a reservation, prompting the controller to forward the request to Room control. Room control will then send a form to user through controller for filling out. Room control does not need to access database therefore cutting down on over head for this part. Once form is filled out, controller will take form and address the input for room control, so room control can now handle retrieving rooms similar to those requested by guest. Room control will then send that info back to controller so it may display it for user to pick and choose. Once chosen, user will initiate final stage wherein, controller will tell room control to book the request and finalize it through database(Database needs to show that room is booked for however long guest is staying).On fulfillment, database will return confirmation through controller.

Use Case 5 - Check In



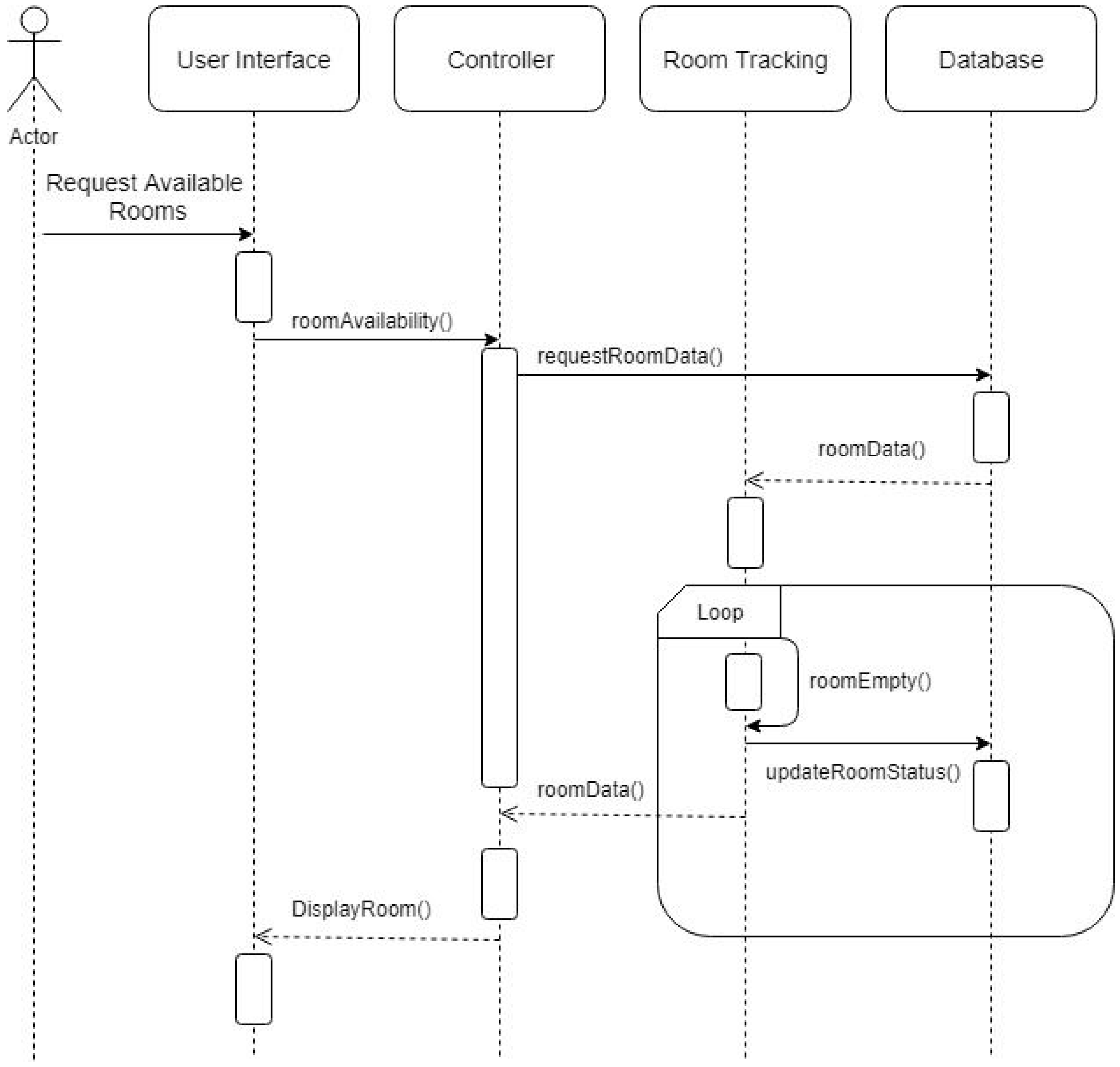
UC-5 refers to how a user can check into their hotel room using our application. It starts off with us assuming user has logged in, clicks check-in button on the user interface. The controller will run the check in function. From here we need to make sure the user has a reservation so the controller will make a request to the database to verify the reservation. The database will confirm the reservation and send it to the room controller so they can provide the user a key for the room and check the user in. After that the control sends the request to display the key on the user interface as well as update the room status in the database.

Use Case 6 - Request Room Service



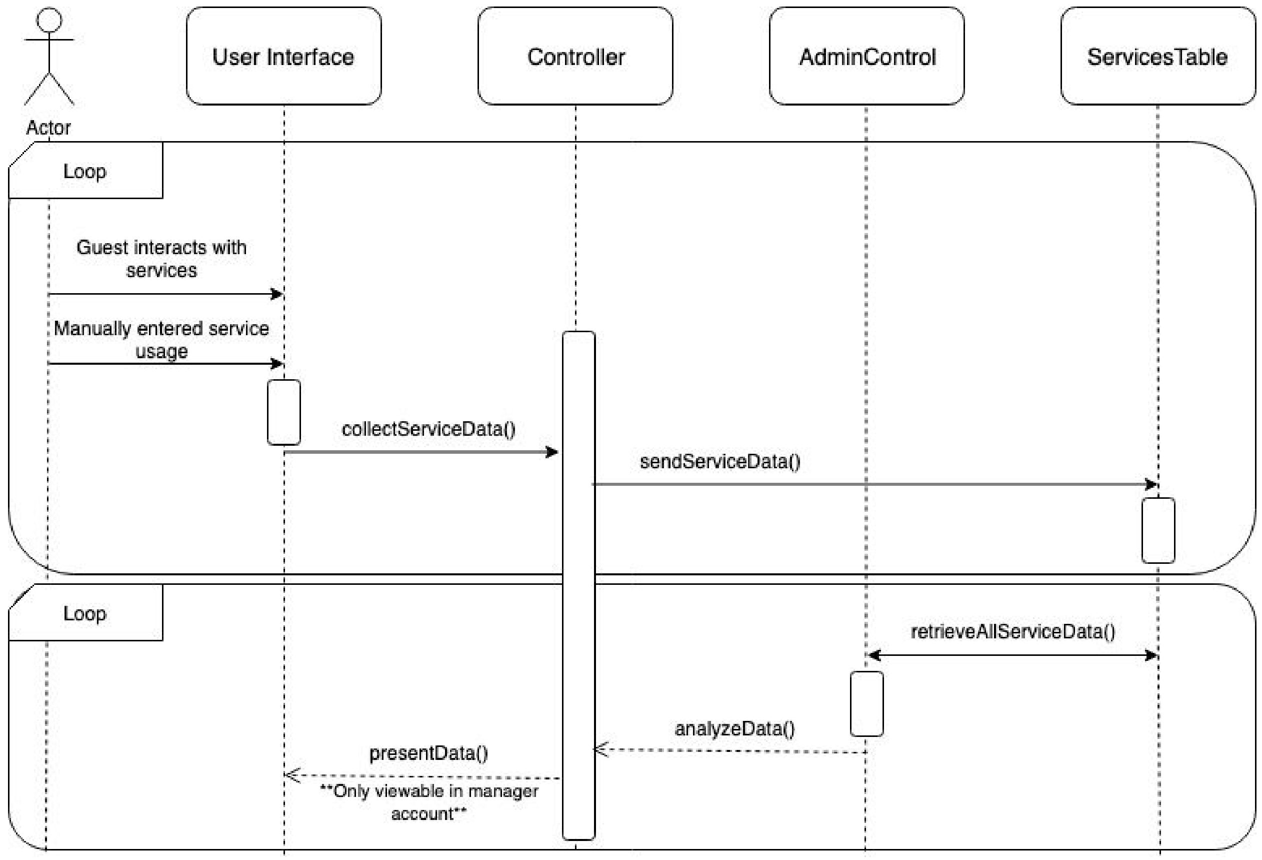
In this use case, the user will be able to call for room service(cleaning, replacing sheets, etc).One of the many functionalities of the app is streamlined requests like this. To access this service, user will input a request from the UI through which controller will receive a signal to Hotel Services which handles these calls. Hotel Services will then forward the request to room attendants to fulfill said request and once request is done, request to mark as done will be passed through controller back to Hotel Services. Once it has been marked as finished, guest will receive a notification of completion as well.

Use Case 7 - Check Availability of Rooms



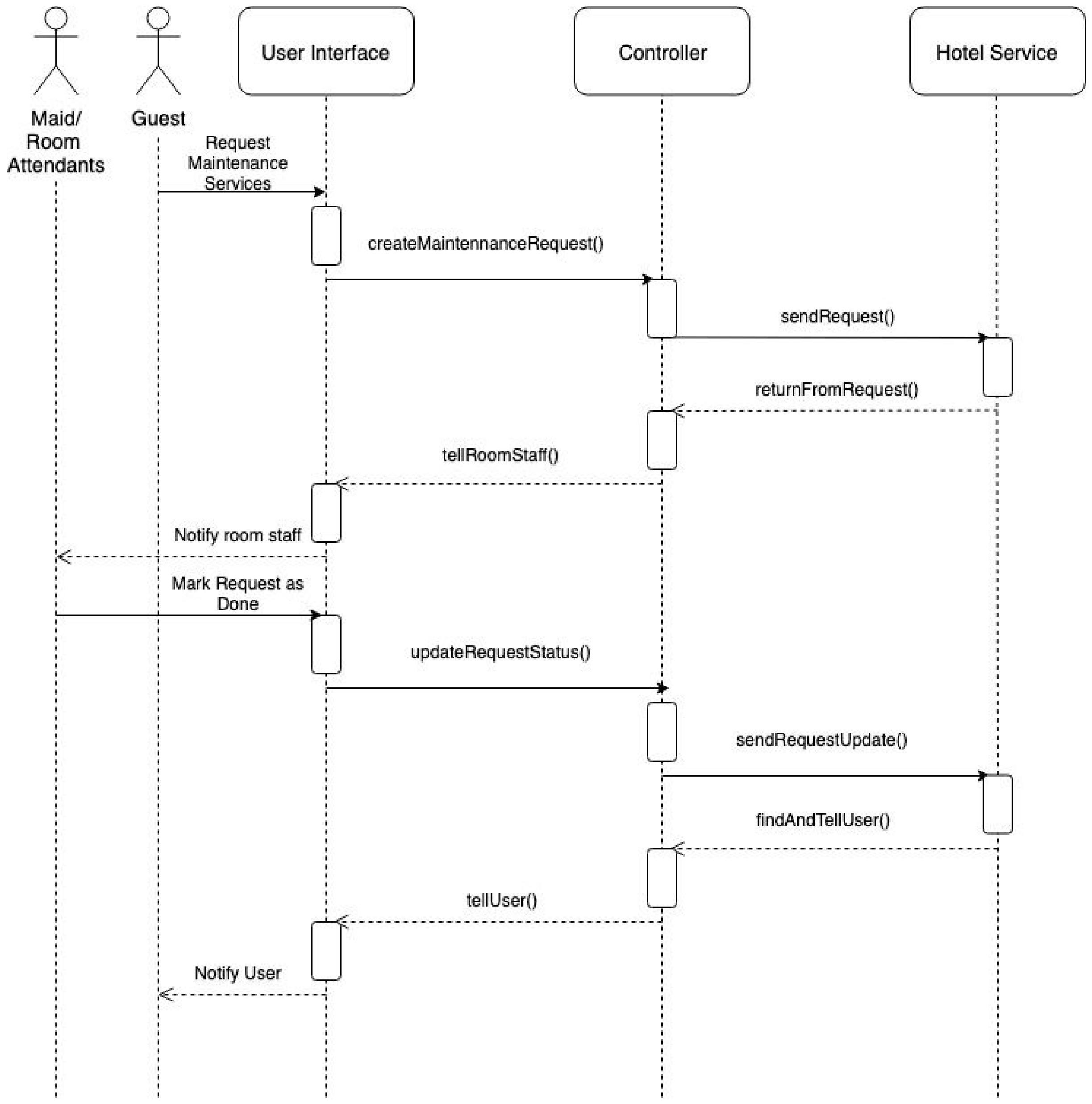
UC-7 allows a user to check the availability of rooms at the hotel. Once they make the request the controller request room data from the database. The database sends this information to the room tracking boundary. Room tracking will run a loop to check if the room is vacant or not. Once the loop runs it will update the status of the room in the database as well as send the data to the controller. The controller then sends the request to display the information on the user interface on whether a room is vacant or not.

Use Case 8 - Review Guest Data



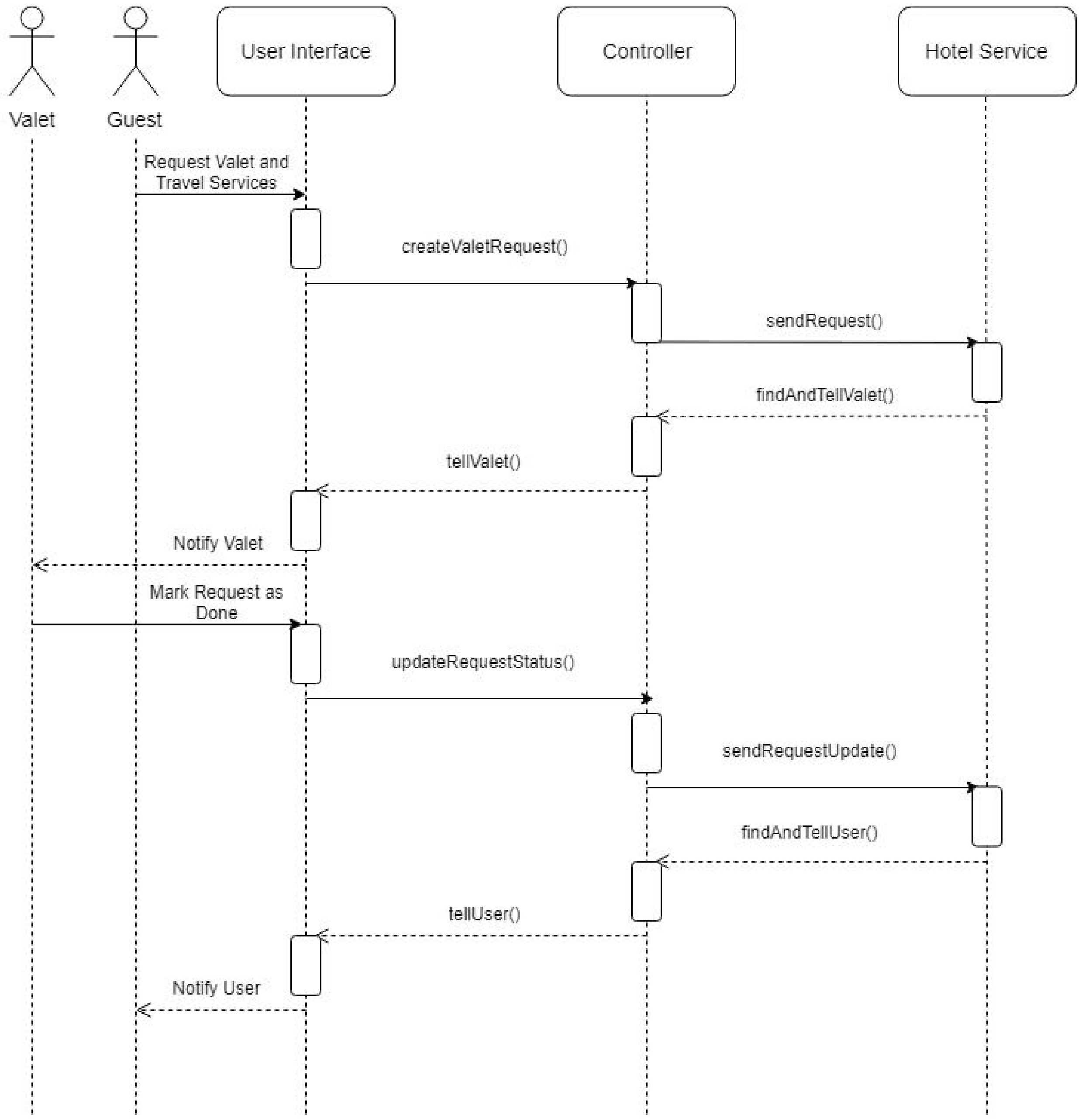
One of the benefits of having an app for hotel automation, is the ability to collect and analyze data. This is a very important use case because it allows administrators to see the usage of the different services provided by the hotel. As guests use the services, or as staff members manually enter usage from guests who do not use the app, the data will be collected by the main controller and sent to the database for storage. Specifically, within the database, we will have a ServicesTable which stores this data along with the frequency and time it was used. The ServicesTable was not in the concepts from the first report because we grouped it together under Database. This process is encapsulated in a loop because as guests continue to use services, the data will be sent and updated in the ServicesTable. The second part of the process involves the AdminControl concept to collect the data from the table. Within AdminControl, we will implement simple data analysis algorithms to be able to present the data back to the administrator. This is also in a loop because as the ServicesTable is updated, the AdminControl must also update the statistics presented.

Use Case 11 - Maintenance Requests



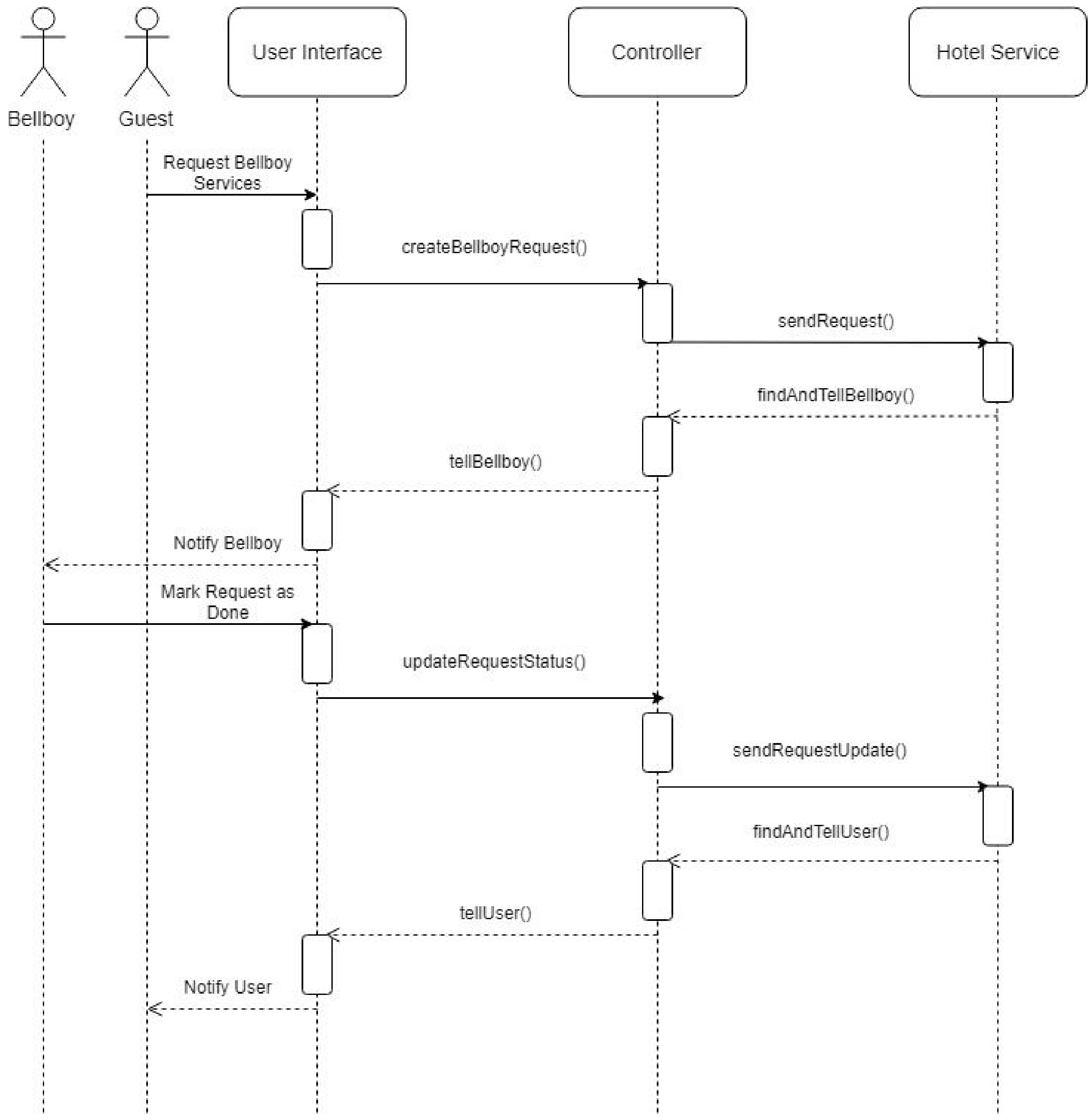
The maintenance requests tab will allow for an efficient way for guests to access request forms for problems within their rooms. Through the app, they will directly be able to navigate to the maintenance request tab, then guests will be able to directly submit their complaint. This is part of the createMaintenanceRequest() function. Consequently, the request will be sent to the database (sendRequest()) and returned to the room staff (tellRoomStaff()) which will allow the room staff to be notified. Once the request has been handled by the room staff and update of the status will be sent through the updateRequestStatus() and then it will be sent to the database. After this, the guest will be informed about the update through the findAndTellUser() and tellUser() functions.

Use Case 12 - Valet & Travel Services



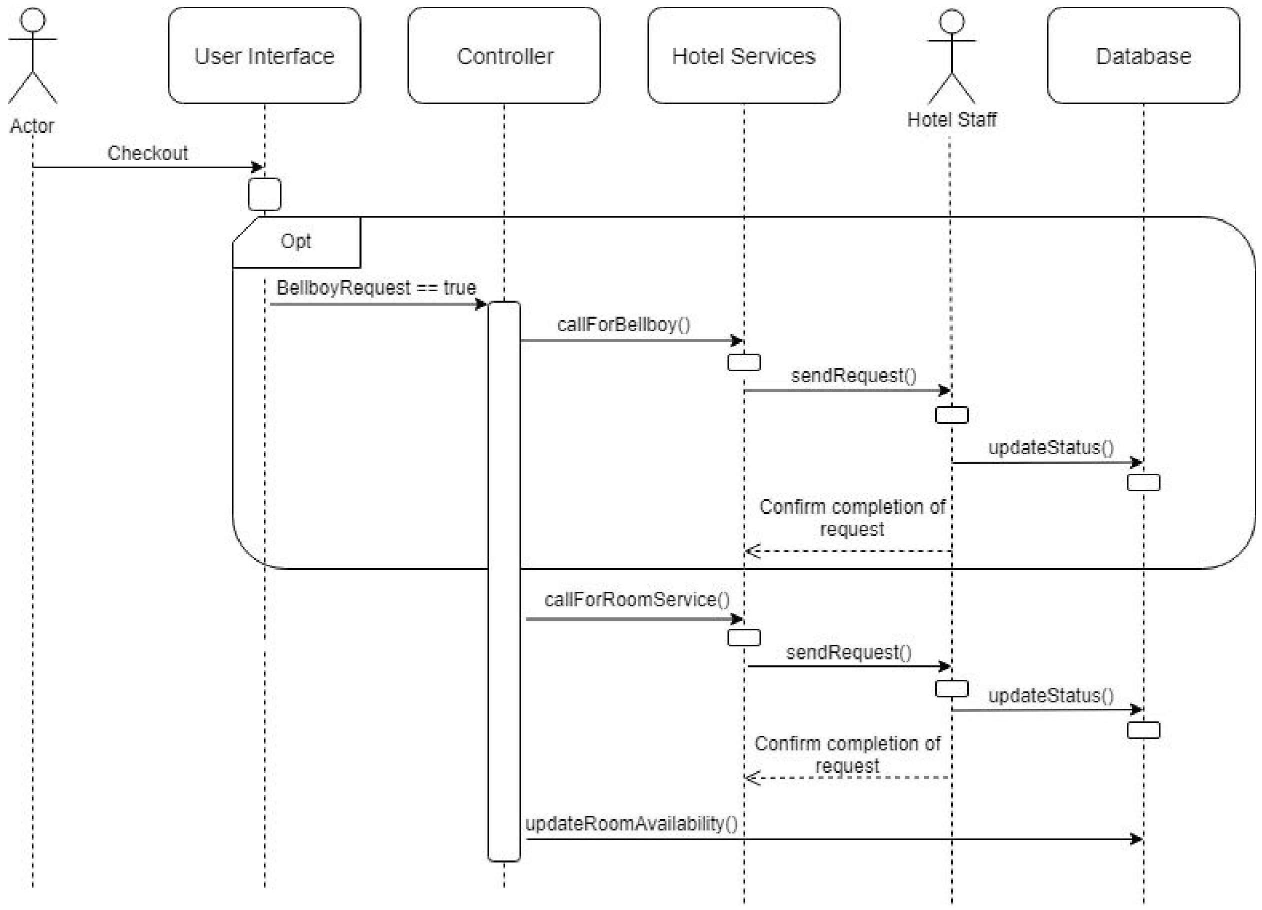
One of the many hotel services includes valet and travel services. This use case is a fundamental service within a hotel. As the guest sends a valet or travel request, the user will specify if they will need to be transported or to have parking services as they submit a request. As staff members, they will receive the guests requests and work to complete the quests requests. They will use the app from the administration side and update the status of the request in the database. The user will receive a confirmation of the complete requested and be notified and will be directed to the home page. This is an outline of how valet and travel services will be processed.

Use Case 15 - Bellboy Requests



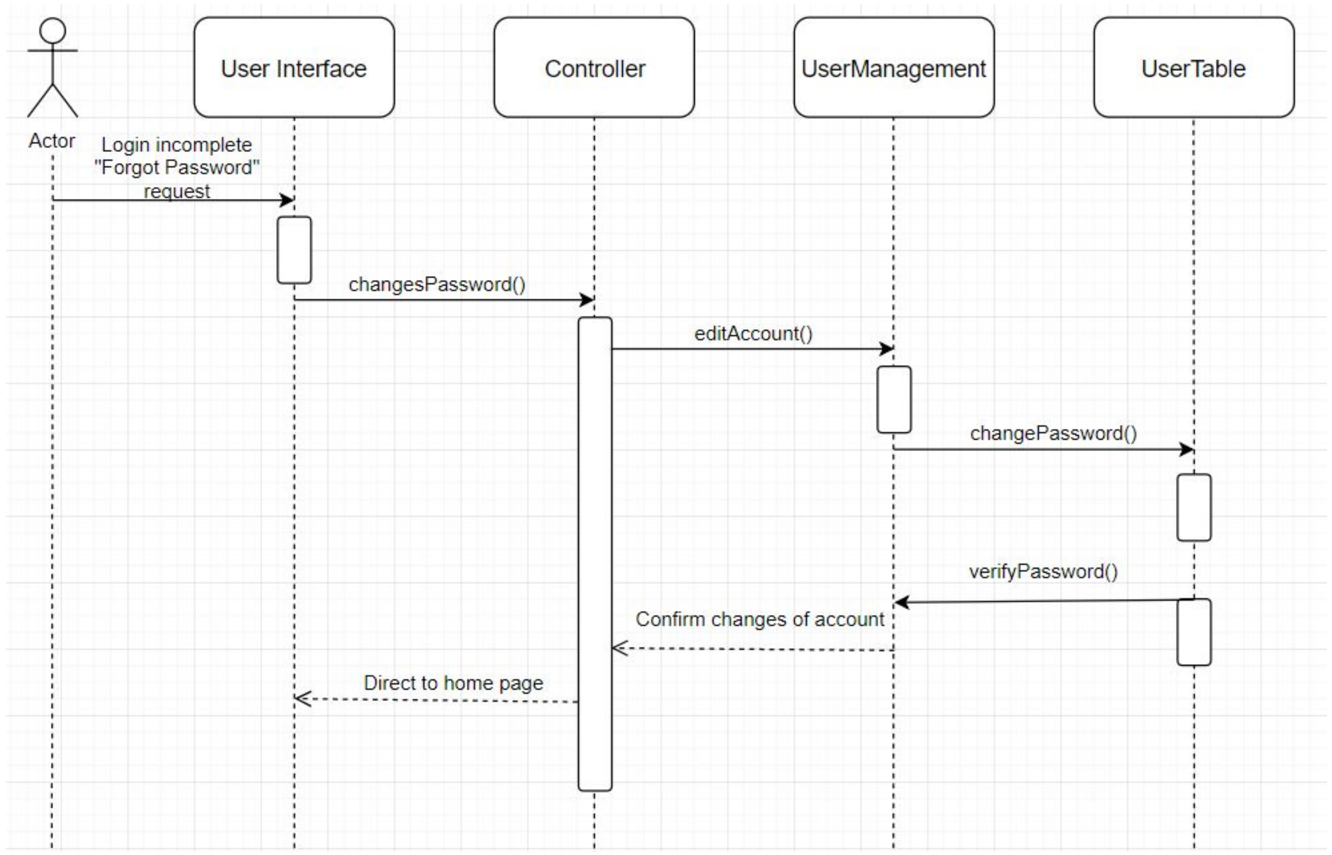
A bellboy system was decided upon to make sure that customers could easily get their luggage taken to their room. The guest will tap on their phones to make a request. This request will be sent to the Bellboy in the form of a notification on his/her app. The bellboy can mark this request as done, and this information will eventually be sent to the guest who will receive a notification that their luggage has been moved to their rooms.

Use Case 16 - Checkout



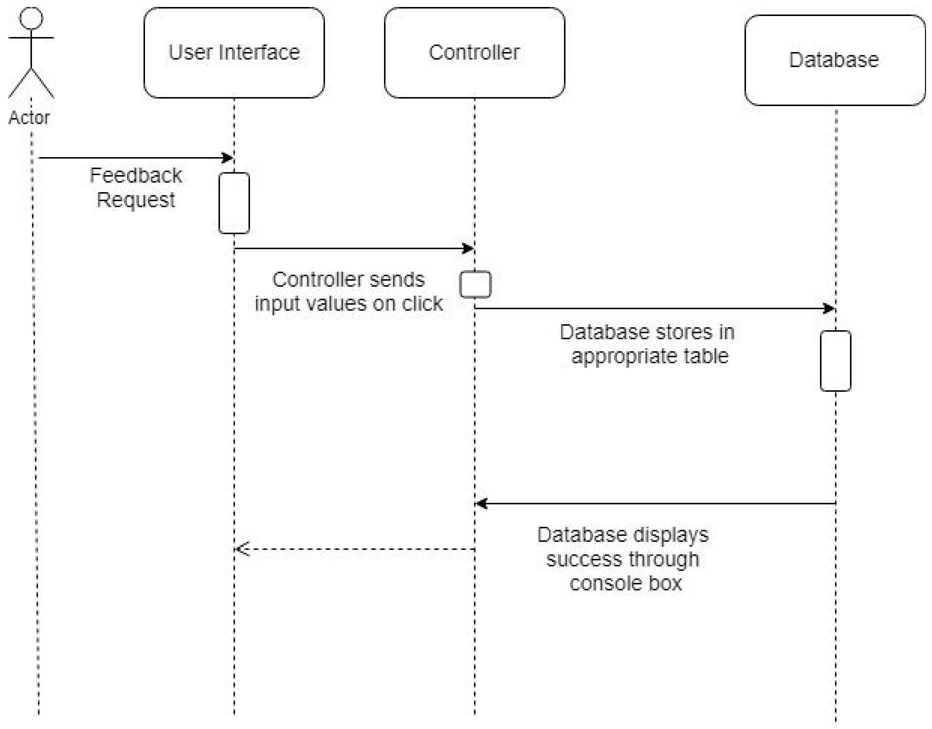
The checkout option through the app is essential to confirm the user has finished their stay at the hotel. After the user has chosen to checkout, they are given the option to request bellboy service to help bring their luggages down to the lobby. On the other hand, room service is automatically notified that they are needed in the room the user has just checked out of. Staff members, bellboys and maids in this case, will use the app to receive their respective requests and to update the statuses of them as well. As soon as the maids have updated the status of this specific request to completed, this will automatically trigger the database to update the number of available rooms.

Use Case 17 - Forgot Password



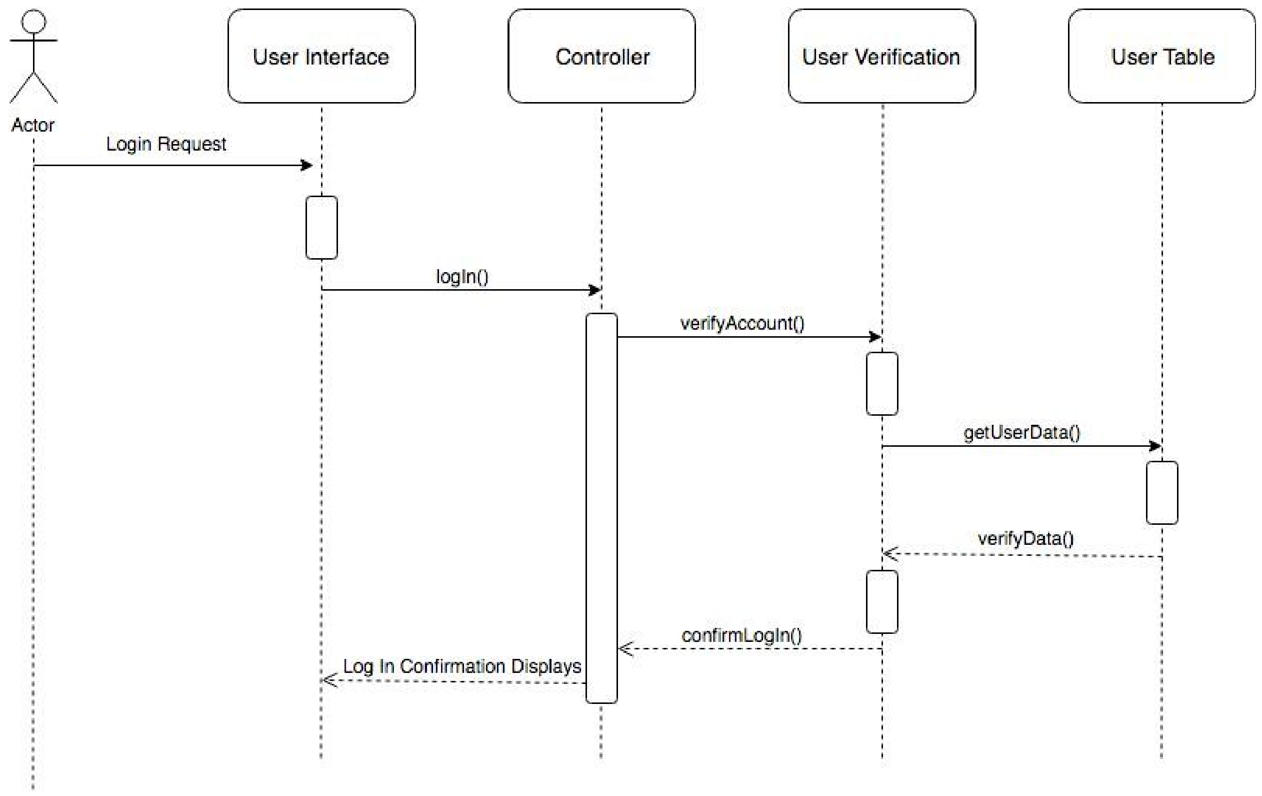
The forgot password option through the app is essential to allow the user to reset their passwords if they forgot it and cannot login into the app. After the user has chosen forgot password option, they are given the option to change their account password through a link we will send. Once the edit is made, the changes will be verified and updated within the database as well. The user should be able to login in will the new password now.

Use Case 18 - Feedback


Actor
User Interface
Controller
Database
Feedback Request
Controller sends input values on click
Database stores in appropriate table
Database displays success through console box

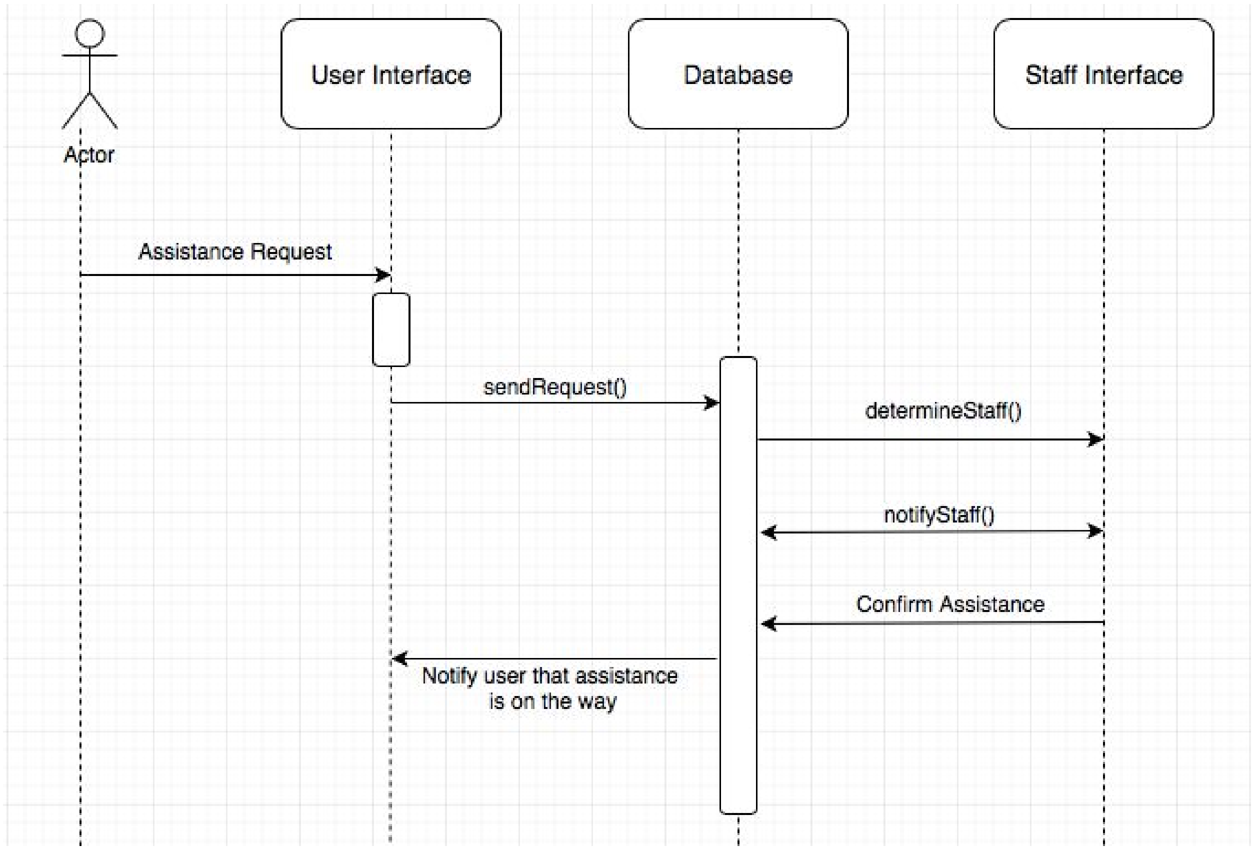
The feedback request is essentially a way for the guest to let administration know how their stay was/is. Using various Star Ratings and Radial Buttons,for example, the guest can let hotel services know if maintenance was great if booking a room was troublesome. Recording this feedback is an important resource because it helps staff pick up where they are lacking as well as recognizing which areas of management are doing well. This process can be described as the following: Feedback button brings user to a page with interactive widgets to help rate hotel services, once user presses submit, controller will send all fields to Firebase and store it in the appropriate table. App will send a console message box on success or failure.

Use Case 19 - Preset Manager Accounts for Hotel Staff



The sequence diagram for preset manager accounts contains the same exact steps as use case 2 - login. However, it is crucial to note that the difference between these two use cases is that manager accounts have access to view service request from guests. Each type of hotel staff has its own respective admin account to view request pertaining to them specifically. Hotel staff is given a preset email and password that allows them to login to “special” accounts for them solely.

Use Case 20 - Send Service Requests to Appropriate Accounts

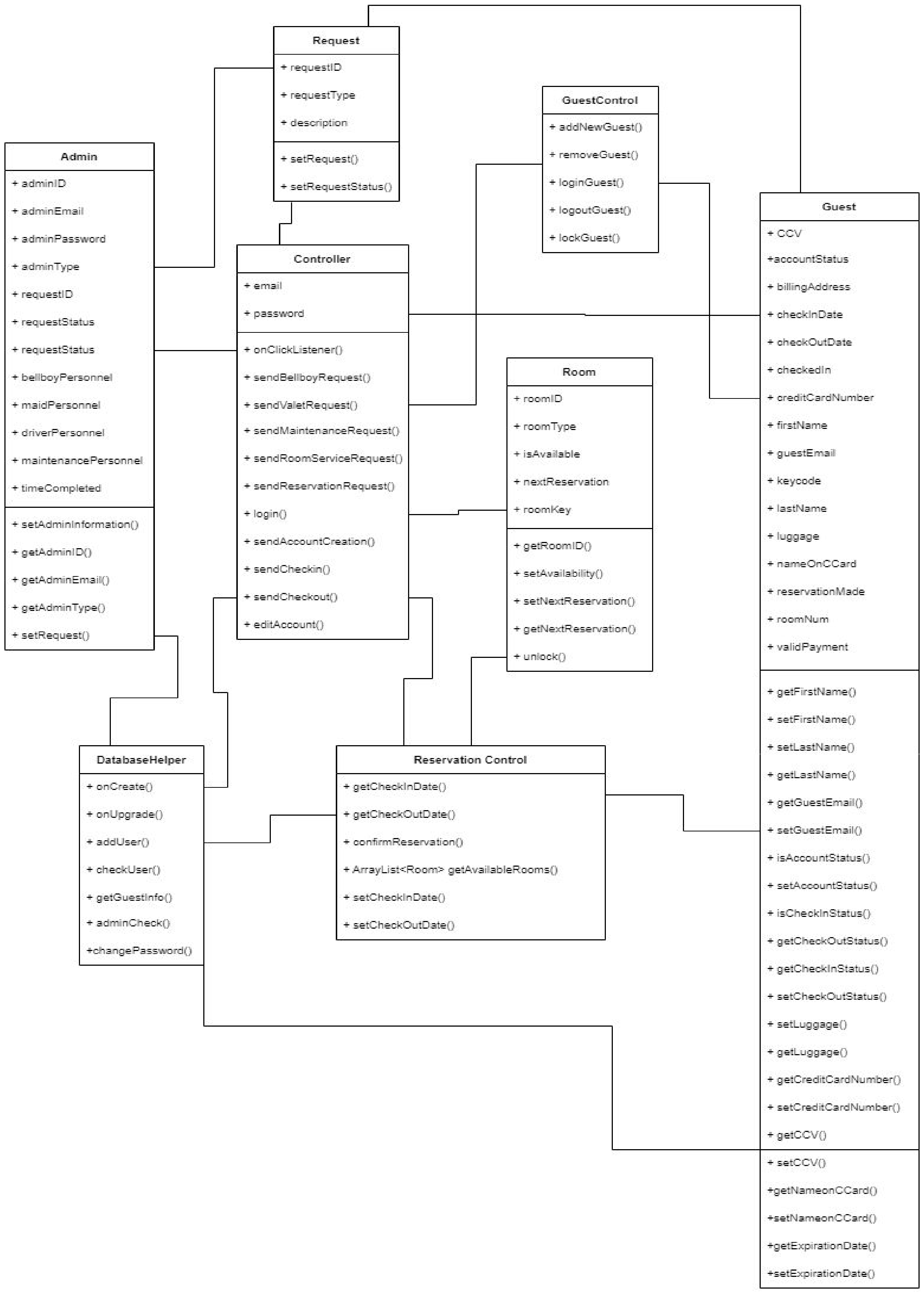


This Use Case sends a request automatically to the staff that is being requested by a guest rather than the admin/manager of the hotel. This makes it faster for the staff to be able to respond to requests from guests and therefore attend to more guests. Once the guest makes the request, depending on what the request is, the database then finds the staff member responsible for that request and sends them a notification as to who needs help, what they need help with and where they are. The staff then is able to accept this service request in order to confirm which goes back to the database and sends a notification to the guest to inform them that assistance is on the way.

Chapter 7

Class Diagrams and Interface Specifications

7.1 Class Diagram


Request
+ requestID
+ requestType
+ description
+ setRequest()
+ setRequestStatus()
GuestControl
+ addNewGuest()
+ removeGuest()
+ loginGuest()
+ logoutGuest()
+ lockGuest()
Admin
+ adminID
+ adminEmail
+ adminPassword
+ adminType
+ requestID
+ requestStatus
+ requestStatus
+ bellboyPersonnel
+ maidPersonnel
+ driverPersonnel
+ maintenancePersonnel
+ timeCompleted
+ setAdminInformation()
+ getAdminID()
+ getAdminEmail()
+ getAdminType()
+ setRequest()
Controller
+ email
+ password
+ onClickListener()
+ sendBellboyRequest()
+ sendValetRequest()
+ sendMaintenanceRequest()
+ sendRoomServiceRequest()
+ sendReservationRequest()
+ login()
+ sendAccountCreation()
+ sendCheckin()
+ sendCheckout()
+ editAccount()
Room
+ roomID
+ roomType
+ isAvailable
+ nextReservation
+ roomKey
+ getRoomID()
+ setAvailability()
+ setNextReservation()
+ getNextReservation()
+ unlock()
Guest
+ CCV
+accountStatus
+ billingAddress
+ checkInDate
+ checkOutDate
+ checkedIn
+ creditCardNumber
+ firstName
+ guestEmail
+ keycode
+ lastName
+ luggage
+ nameOnCCard
+ reservationMade
+ roomNum
+ validPayment
+ getFirstName()
+ setFirstName()
+ setLastName()
+ getLastName()
+ getGuestEmail()
+ setGuestEmail()
+ isAccountStatus()
+ setAccountStatus()
+ isCheckInStatus()
+ getCheckOutStatus()
+ getCheckInStatus()
+ setCheckOutStatus()
+ setLuggage()
+ getLuggage()
+ getCreditCardNumber()
+ setCreditCardNumber()
+ getCCV()
+ setCCV()
+getNameonCCard()
+setNameonCCard()
+getExpirationDate()
+setExpirationDate()
DatabaseHelper
+ onCreate()
+ onUpgrade()
+ addUser()
+ checkUser()
+ getGuestInfo()
+ adminCheck()
+changePassword()
Reservation Control
+ getCheckInDate()
+ getCheckOutDate()
+ confirmReservation()
+ ArrayList<Room> getAvailableRooms()
+ setCheckInDate()
+ setCheckOutDate()

7.2 Class Data Types and Operation Signatures

Controller

The controller coordinates actions of all concepts associated with the use cases and logically groups the subsystems to work together. It is the main interface between the system and the application that the user sees.
The controller will be handling all user interactions with the app, such as button clicks and entering text information. It is also responsible for transitioning XML pages as the app is event-controlled and the user can navigate to multiple pages.

Methods:

1) On-click listeners for all buttons
    a) Will trigger different pages in the UI and button clicks may cause any of the below methods to be called
2) sendBellboyRequest()
    a) Will call the Hotel Services Controller
3) sendValetRequest()
    a) Will call the Hotel Services Controller
4) sendMaintenanceRequest()
    a) Will call the Hotel Services Controller
5) sendRoomServiceRequest()
    a) Will call the Hotel Services Controller
6) sendReservationRequest()
    a) Will call the Reservation Controller
7) login(string email, string password)
    a) Will call the Guest Controller and allow access to rest of app
8) sendAccountCreation()
    a) Will call the Guest Controller and allow user to login
9) sendCheckin()
    a) Will call the Guest Controller
10) sendCheckout()
    a) Will call the Guest Controller

Guest

Attributes:

1) int guestId

    a) Unique ID number given to each guest

2) String userEmail
    a) Specific email address of guest to login
3) String firstName
    a) First name of guest
4) String lastName
    a) Last name of guest
5) Date accountCreation
    a) Stores when exactly the account was created
6) String accountStatus
    a) Determines the status of the account (active, inactive, locked etc)
7) int luggage
    a) Determines whether the guest has specified if they have luggage or not
    b) Information used by Bellboy service
8) boolean hasReservation
    a) Boolean value to see if guest has made a reservation for a room, in which case they may check in upon arrival
9) boolean checkedIn
    a) A boolean value to see if guest is checked in or not
    b) Checked-in status confirms that the guest is at the hotel currently
    c) Going from a checked-in equals "true" status to a checked-in equals "false" status indicates that the guest has finished their stay and have now checked out
10) String checkInDate
    a) Specifies the check in date of the guest if they made a reservation
11) String checkOutDate
    a) Specifies the check out date of the guest
12) String creditCardNumber
    a) Stores credit card info
13) String billingAddress
    a) Stores credit card info
14) String expirationDate
    a) Stores credit card info
15) Boolean validPayment
    a) Stores credit card info
16) Boolean reservationMade
    a) Determines if reservation is made (but the guest has not checked in)
17) String keycode
    a) Integer value with a unique room key ID number for guests to unlock their room doors

18) int requestId
    a) Unique ID number pertaining to a new request a guest has made

Methods:

1) void Guest(String firstName, String lastName, String guestEmail)
    a) Creates new guest object with defined parameters
2) String getKeyCode()
    a) Get guest room key information based on guest information
3) void setAccountStatus(String status, boolean loggedIn)
    a) Sets account status of account corresponding to guest ID (active, inactive, locked, etc)
    b) Sets loggedIn variable of account
4) String getAccountStatus()
    a) Gets account status of account corresponding to guest ID (if user is logged in, if account is active, inactive, or locked)
5) void setCheckedIn(boolean hasReservation)
    a) Checks hasReservation variable, then sets checkedIn variable
6) Boolean getCheckedIn()
    a) Gets check-in status of account corresponding to guest ID
7) void setLuggage(int luggage)
    a) Sets number of luggage pieces user is bringing
8) int getLuggage()
    a) Gets number of luggage pieces user is bringing

Note: Each field has a getter and setter method but most are self explanatory based on the variable names

GuestControl - Interacts directly with main controller for functionality

Methods

1) addNewGuest()
    a) Add new guest to database
2) removeGuest(int guestId)
    a) Remove a guest from database, specified by ID number
3) loginGuest(int guestId)
    a) Mark boolean value to true if guest is successfully logged in
4) logoutGuest(guestId)
    a) Mark boolean value to false if guest logs out of their account
5) lockGuest(guestId)
    a) Mark boolean value to true if guest has been locked from their account

RequestObject

Attributes:

Common Core Attributes:

1) Private long requestID
    a) Identifying number of a new request
2) PrivateString requestType
    a) String value that contains the type of request a guest has made (“bellboy”, “maintenance”, “room service”, or “travel”)
3) Private String inputs
    a) String value containing a description of the guest’s problems or information/specifics of the request
4) Private String requestedTime
    a) String that details the time at which the guest wishes the service to be performed
5) Private String requestedDate
    a) String that details the date that the guest wishes the service to be performed

List of all attributes:

6) private String description;
7) private String status;
8) private String hourValue;
9) private String minuteValue;
10) private String ampmValue;
11) private String luggageValue;
12) private String requestedTimeValet;
13) private String requestedTimeBellboy;
14) private String requestDate;
15) private String inputs;
16) private String bathroom;
17) private String electronic;
18) private String lighting;
19) private String checkboxes;
20) private String towels;
21) private String soap;
22) private String bedsheets;
23) private String cleaningservice;
24) private String requestedTimeMaintenance;
25) private String requestedTimeRoomService;
26) private String fromWhere;
27) private String startingStreet;

28) private String startingCityStateZip;
29) private String destinationStreet;
30) private String destinationCityStateZip;
31) private String numberTraveling;
32) private String temp1;
33) private String temp2;
34) private String requestedTimeFoodService;
35) private String gardenSalad;
36) private String tomatoSoup;
37) private String friedChicken;
38) private String cheesePizza;
39) private String spaghetti;
40) private String macAndCheese;
41) private String vanillaIceCream;
42) private String fruitCake;
43) private String coke;
44) private String sprite;
45) private String appleJuice;
46) private String foodPrice;
47) private Integer a1;
48) private Integer a2;
49) private Integer m1;
50) private Integer m2;
51) private Integer m3;
52) private Integer m4;
53) private Integer d1;
54) private Integer d2;
55) private Integer dr1;
56) private Integer dr2;
57) private Integer dr3;

Methods:

1) Getter and Setter for each attribute to get and set values into objects created in the service activities to pass into the database as a JSON object.

Room Object

Attributes:

1) int roomId
    a) Unique number assigned to each room

2) String roomType
    a) “Single”, “Double”, “Queen”, or “King”
3) boolean isAvailable
    a) Flag to indicate if the room is occupied or not
4) Date checkInDate
    a) Date object to indicate when the room is reserved
5) Date checkOutDate
    a) Date object to indicate when the room is available

<u>Methods:</u>

1) int getRoomId()
    a) Returns the unique number of the room
2) void setAvailability(boolean isAvailable)
    a) Method to set the availability to of the room to true or false
3) void setNextReservation(Date nextReservation)
    a) Method to set the reservation of the room if a new request was made successfully
4) Date getNextReservation()
    a) Method to get the next reservation date of the room
5) boolean unlock(int roomKey)
    a) Method to compare if user entered room key matches with the room’s actual key value, will return true if unlocked and false if the key do not match, leaving the door locked

ReservationControl - Implements functionality to create new reservations

Methods

1) getReservationDates()
    a) Gets information entered by guest in the UI
2) ArrayList<Room> getAvailableRooms(Date startDate, Date endDate)
    a) Based on the duration of stay, check all the rooms that are available and return an ArrayList of these Room objects
    b) The controller will output this ArrayList to the UI so the guest can see the available rooms
3) confirmReservation(Room bookedRoom)
    a) When the guest selects a room, send that object back to mark the room as reserved for those dates

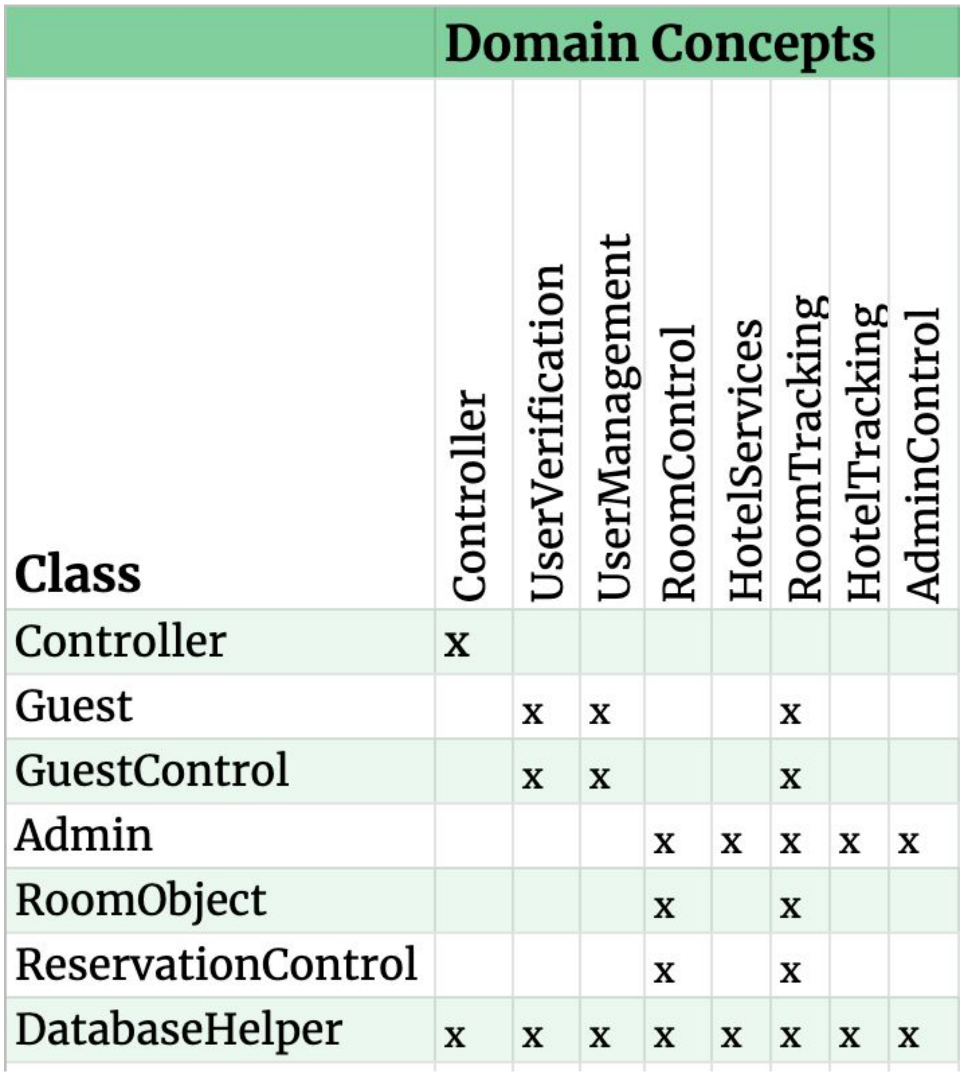
DatabaseHelper

Main object that manages the database objects and all the methods that directly input and extract data from tables. As we continue implementing functionality, the DatabaseHelper methods will continue to increase. Maintaining database information is the backbone of our app because all the data is used to help serve the guest.

<u>Methods</u>

1) onCreate()
    a) Default by Android studio
2) onUpgrade()
    a) Default by Android studio
3) addUser()
    a) Adds a new user to the table
4) checkUser()
    a) Checks if a user exists in the table
5) adminCheck()
    a) Checks if an admin login is valid.
6) getGuestInfo(int guestId)
    a) Returns a Guest object from the given ID number

7.3 Traceability Matrix



	Domain Concepts							
Class	Controller	UserVerification	UserManagement	RoomControl	HotelServices	RoomTracking	HotelTracking	AdminControl
Controller	x							
Guest		x	x			x		
GuestControl		x	x			x		
Admin				x	x	x	x	x
RoomObject				x		x		
ReservationControl				x		x		
DatabaseHelper	x	x	x	x	x	x	x	x

The **Controller** class will be the interconnected with all other classes. This is the only domain concept that's itself because everything is directly accessible through this class.

The **Guest** object will keep track guest account information on luggage status, reservations, check in/out, and account status. This is related to UserVerification because it is needed to check with the database if the email has already been used before. It is also part of the UserManagement concept because all their account information and object fields will be stored in the database.

The **GuestControl** will be responsible for checking proper login request for each guest and marking them as currently logged in and out. Additionally, GuestControl will also be in charge of adding and removing guests from the database.
The **Admin** object will be directly from the domain concepts AdminControl but also interacts with other concepts. This will be the platform in which hotel admin will be able to login to their own system. This falls under the AdminControl as the administrators/hotel staff (i.e bellboy, valet) will all be the same object, but differentiated by their own separate duties/roles within the system. This distinction between guest and administrator will be necessary for functions within the app.

The **RoomObject** will be related to RoomTracking, and RoomControl. Each Room will be its own object once the guest checks into the room. The object will be used to manage the different parts of the room, such as room key, maintenance request and reversed dates. RoomTracking concept is meant to analyze all the types of rooms available on specific dates, so this will use the Room object data to accomplish this. And the RoomControl concept allows the guest to make a reservation and check-in, so the Room object data will be updated accordingly.

The **ReservationControl** is for retrieving information of reservations from guests and identifying the available rooms for available days. This interacts with the RoomControl and RoomTracking to determine the available rooms at the time of the request. Once the reservation is confirmed, the data is sent back and updated in the concepts.

The **DatabaseHelper** is a separate object to handle all the SQL statements and directly change the stored data. Within this class, there will be methods for each object to interact with, for example, having an addUser() methods to run SQL that add a user to the UserTable. Therefore, the DatabaseHelper will interact with each concept because each concept requires data management.

7.4 Design Patterns

The main functionality of the app revolves communication between the guest and admin side. The guest side features the ability to perform tasks such as creating an account, making a reservation, checking in, making service requests, among other things. When the user navigates to a page, the guest will be able to input information based on directions in the page. For example, in the bellboy service request, the user is prompted for information regarding the time and date for when he wants the service, as well as the number of luggage to be carried and from where they would liked their bags to be picked up. After the user fills in his information, on clicking the submit button the data is recorded from the elements in the xml file and recorded in the activity java file, which is then stored in a service object and sent to the database as a JSON. From here, the admin side can pull data from the database based on the requests, and view them or change their progress statuses. As mentioned earlier, this model of having the guest fill in the application page, sending it to the database as an object, and then being able to view the object's data from the database on the admin side is a typical design pattern used throughout most functionalities of the application. This pattern serves a critical function of providing a way for the guests to communicate with the hotel staff, and vice versa, which is the main purpose of this app.

7.5 Object Constraint Language

Register an account

context RegisterActivity:

Pre: regPassInput = input

regPassInput.length() > 6

Post: task.isSuccessful() == true //added account to firebase

RegisterActivity->LoginActivity

Login to account

context LoginActivity:

Pre: passInput.empty() = false

Post: task.isSuccessful() == true // logged in

LoginActivity->HomeActivity

Logout

context HomeActivity:

Pre:

Post: HomeActivity->MainAcivity

Make a reservation

Context ReservationActivity

Pre:

Post: ReservationActivity->newViewRooms

Check-in

Context CheckInActivity

Pre: currentDate.equals(chekInDate)

Post: updateCheckIn() //sets checkedin on database

CheckInActivity->HomeActivity

Room Service

Context RoomServiceActivity

Pre: checkedIn == true;

Post: task.isSuccessful() //update database with request

RoomService->ServicesActivity

Check Availabilities of Rooms

Context newViewRooms

Pre:
Post:

Review Guest/Service Data
Context AdminActivity
Pre: admin == true
Post:
AdminActivity-> AdminServiceList
Admin Activty-> ViewGuests

Request Status
Context AdminServiceList
Pre: //logged into admin
Post: //each page will display service status
AdminServiceList ->BellboyServices
AdminServiceList ->ValetServices
AdminServiceList ->MaintenanceServices
AdminServiceList ->RoomServices

Maintenance
Context MaintenanceActivity
Pre:
Post: task.isSuccessful()//update database with request
MaintenanceActivity->ServicesActivity

Valet & Travel Services
Context ValetTravelActivity
Pre:
Post: task.isSuccessful()//update database with request
VaeltTravelActivity->ServicesActivity

Feedback
Context FeedbackActivity
Pre:
Post: task.isSuccessful()//update database with feedback

Hotel Information
Context MainActivity

Pre:
Post: task.isSuccessful()
MainActivity>HotelInfo

Bellboy Services
Context BellboyActivity
Pre:
Post: task.isSuccessful() //update database with request
BellboyActivity->ServicesActivity

Checkout
Context HomeActivity
Pre:
Post:
HomeActivity->MainActivity

Forgot Password
Context PasswordActivity
Pre:
Post: task.isSuccessful()//update password in database
PasswordActivity->MainActivity

Feedback For Admin

Preset Manager accounts for hotel staff
Context LoginActivity
Pre:
email.equals("admin@mail.com"))
email.equals("bellboyadmin@mail.com")
email.equals("valetadmin@mail.com")
email.equals("maintenanceadmin@mail.com")
email.equals("roomservicesadmin@mail.com")
Post:
task.isSuccessful()//update password in database
LoginActivity>AdminActivity
LoginActivity>BellboyAdmin
LoginActivity>ValetAdmin
LoginActivity>MaintenanceAdmin

LoginActivity>RoomServiceAdmin

Chapter 8

System Architecture

8.1 Architectural Styles

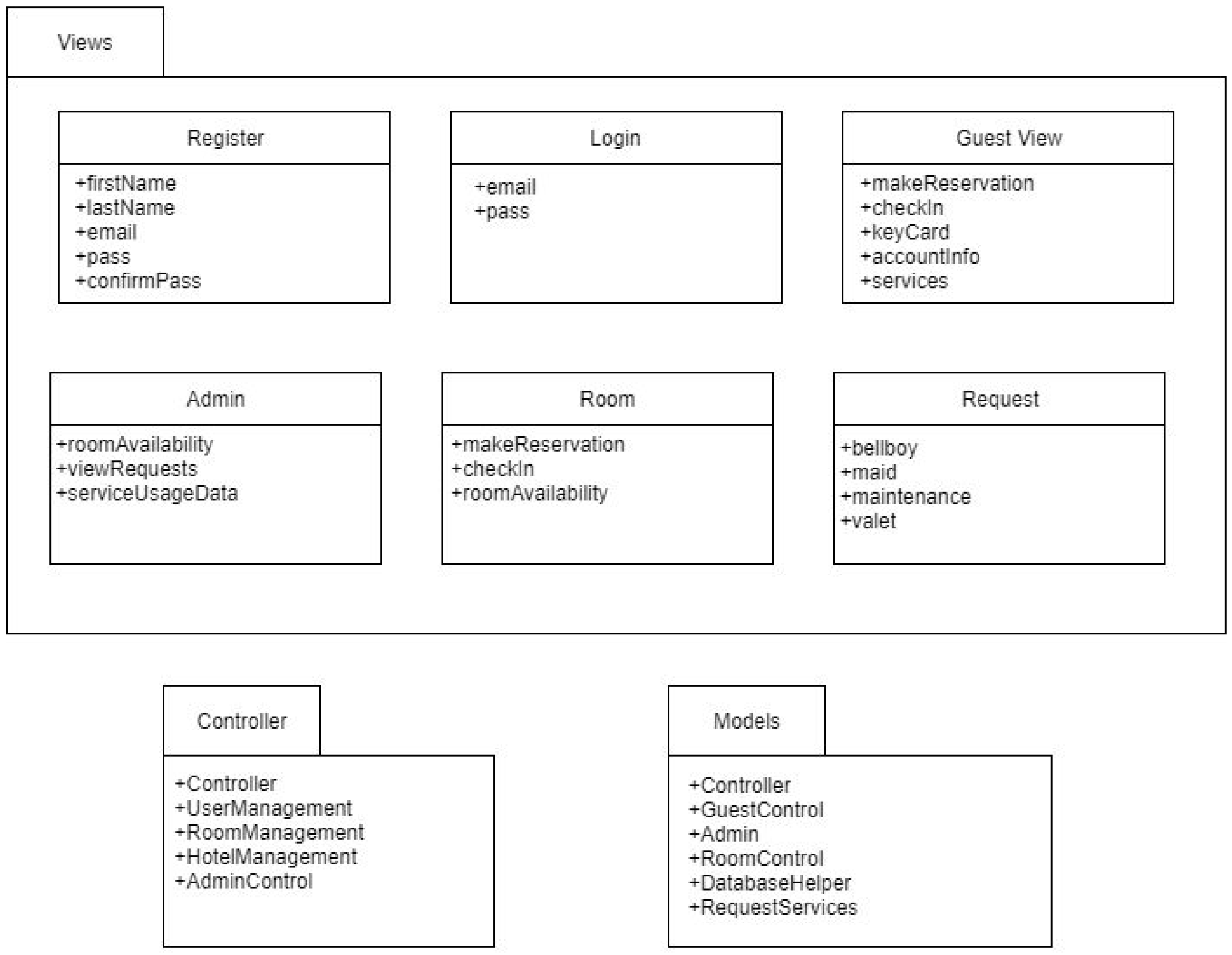
Communication
The basis for our RUstaying app lies upon the many functionalities that we provide to our guests through hotel automation. Through Service-Oriented Architecture we are building each of these functions independent of each other. These implemented services communicate messages with our app through our formally defined XML schemas in Android Studio. This allows us to reuse code and simplify the management of said code. SOA helps keep our code well-defined and self-contained with its various functions because at the end, we need robust capability for the app to be able to communicate with an end user and other entities like databases or another user with admin privileges. An advantage of using this architecture is that if we were to scale up or scale down on functions, the rest of the functions would still be just as efficient since they are self-reliant.

Deployment
Current version of app uses Firebase. For the deployment of our app, once completely implemented, ideally we would need a database which is accessible through a cloud for when the Model needs to load/store data. This type of interface which is essentially a client-service communication so as a result we are using a client-server model. When there are multiple platforms supporting the app, such functionality for the interface needs to be consistent across all devices so having this model gives us the most efficiency. Ultimately, we rely on these user-server requests and this model gives us the most efficiency and scalability

Structure
Using MVC(Model-View-Controller) for Object Oriented Architecture
In our case, the controller component is what initiates a function, manipulates data in the Model, and returns some values and displays the data for end user to view. Generically, once the user has some input in the UI, the controller will handle the logic and call the function requested. The Model, which in this case are the independent functions(may or may not use database) will load/store data. Here is where all the classes which the RUstaying team has written will come into play. Once controller has received confirmation that the function has completed

partially/completed (app might need more input from UI for complex functions), controller will ask View to update for further instructions.

8.2 Identifying Subsystems



8.3 Persistent Data Storage

In order to keep track of data for long periods of time, we must use persistent data storage. We are using Firebase to store and track data obtained from the user. It is a NoSQL Cloud database management system. Data is synced in real time across all clients, and always remains available even when the app goes offline. Data is stored as a JSON and synchronized in real time to every connected client. When adding data to the JSON tree, each entry becomes a node in the existing JSON structure with an associated key.

To store persistent data, we will be creating objects with key values to have and use these objects to pass data into the JSON tree. We will have objects such as Guest, Room, and Services. Here is short description of each object and its use of persistent data:

```
Public Class ‘Guest’(
   private String firstName, lastName, guestEmail;
   private String accountCreated;
   private boolean accountStatus;
   private int luggage;
   private boolean isCheckedIn;
   String roomNum;
   private String checkInDate;
   private String checkOutDate;
   private String creditCardNumber;
   private String CCV;
   private String nameOnCCard;
   private String billingAddress;
   private String expirationDate;
   private boolean validPayment;
   private boolean reservationMade;
)
```

Description: As guestID, userPassword, and userEmail is required to login, this information must be persistent. The system should also keep track of the user first and last name. accountCreation, accountStatus, hasReservation, checkIn, loggedIn are also all important fields to keep persistent as some of these booleans need to be TRUE for the user to activate certain services. roomKeyID should most definitely be persistent as the user would not be able to enter the room without it each time.

```
Public Class ‘Room’(
        ‘roomId’ INT,
        ‘roomType’ TEXT,
        ‘isAvailable’ BOOLEAN,
        ‘nextReservation’ DATE,
        ‘roomService’ TEXT,
        ‘maintenanceRequest’ TEXT
)
```

Description: To keep track of each room and its availability, it is vital that we keep fields such as roomId, roomType, isAvailable and nextReservation persistent. Without this persistent data, the staff would not know which rooms are currently available or will be in the future.

```
public class Service {
   `requestID`LONG
   `requestType`TEXT,
   `description`TEXT,
   `status`TEXT,
   `hourValue`TEXT,
  `minuteValue`TEXT,
  ` ampmValue`TEXT,
   `luggageValue`TEXT,
   `requestedTimeValet`TEXT,
   `requestedTimeBellboy`TEXT,
   `requestDate`DATE,
   `inputs`TEXT,
   `bathroom`TEXT,
   `electronic`TEXT,
   `lighting`TEXT,
   `checkboxes`TEXT,
   `owels`TEXT,
   `soap`TEXT,
   `bedsheets`TEXT,
    `cleaningservice`TEXT,
   `requestedTimeMaintenance`TEXT,
   `requestedTimeRoomService`TEXT,
   `fromWhere`TEXT,
   `startingStreet`TEXT,
   `startingCityStateZip`TEXT,
  `destinationStreet`TEXT,
  `destinationCityStateZip`TEXT,
  `numberTraveling`TEXT,
   `temp1`TEXT,
   `temp2`TEXT,
   `requestedTimeFoodService`TEXT,
   `gardenSalad`TEXT,
   `tomatoSoup`TEXT,
   `friedChicken`TEXT,
  `cheesePizza`TEXT,
   `spaghetti`TEXT,
  ` macAndCheese`TEXT,
   `vanillaIceCream`TEXT,
   `fruitCake`TEXT,
   `coke`TEXT,
   `sprite`TEXT,
   `appleJuice`TEXT,
   `foodPrice`TEXT,
}
```

Description: To keep track of each submitted service, and then in separate subclasses use overloading to specify the type of each service. For example: bellboy requests are sent to this subclass:

```
public Service {
     ‘  requestType’ TEXT
     ‘ requestDate ‘TEXT
     ‘ luggageValue’TEXT
      ‘requestedTimeBellboy ‘TEXT
     ‘ fromWhere’TEXT
     ‘ status’TEXT
  }
```

This ensures that we do not need to populate each request with 50-60 different attributes each time, and keeps the request types separate. This same technique is used to valet, maintenance, room service, and food services.

8.4 Global Control Flow

Execution Order

This app is procedure-driven up until after registration and login, meaning every user will go through the same steps every time. The rest following is event-driven which means the user determines what the app is used for and which features they want to use. Upon opening the app, all users must either log in or if they do not have an account, register and then log in. Each user can use the app whenever they want and in order of whatever features they would like after being logged in. For example, once the user is logged in, they can choose to do anything like book a room, request a service, or even logout.

Time Dependency

This app is a real-time dependent system that refreshes its information every 5 seconds so that the user and admin can both have the most updated version on room vacancies, check outs, check ins and if a maintenance/room service request has been completed. Every function in the app is either dependent on what the users or hotel staff is doing at any given moment. For example, a guest will check into a room for a set amount of days in real time and hotel staff will be completing services such as room service in real time and the app will be updated.

Concurrency

Our app will not be using multithreading or multiprocessing. The user interface is event-controlled and each action only requires linear execution. There is no need for concurrency because the user can only initiate one action at a time. Synchronization is not directly used within our app but will be used by the database to store information properly. Because multiple users can be using the app at the same time, there could be multiple requests to get or update the data stored in the database. This, however, is managed directly by Firebase in its implementation. We will not need to worry about synchronization because Firebase will be managing concurrent data requests at the same time to ensure the data is not corrupted.

8.5 Hardware Requirements

As a whole, the majority of computational resources are used on the server-side with the use of Android Studio, Firebase database, and a Java backend which can be accessed by Android phones. The system relies on the use of internet connection in order to constantly pull data and verify the features of the application the user chooses to the admin and the employees of the hotel. This is also necessary in order to maintain consistency of uses for all of the users and maintain a constant updated version of room status, service status, check out/check in, etc. The internet is also required for the concierge feature of the app in order to help the user find nearby activities/restaurants or allow any questions of the user to be sent to a concierge themselves to answer. Due to the fact that we are using Android Studio, we are able to customize the app's layout, allowing us to be flexible with the resolution, scaling size and disk space. Android Studio allows us to create alternate layouts as well in order to adjust to how the user is holding their phone while viewing the app- horizontally or vertically. Users will be able to access the app through most Android smartphones or simulators that act as one.

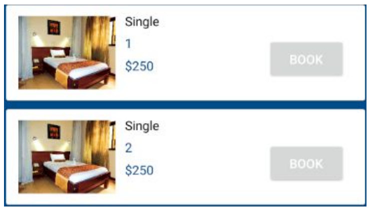
Chapter 9

Algorithms and Data Structures

Algorithms

The only algorithms that we used in this application was a basic pricing algorithm used to display the current prices of our rooms depending on the date of the stay. In our fictional luxury Varghese Hotel, we offer room types such as single, double, queen, or king sized rooms. We initialized each type with a base price, and then researching peak hotel activities in other luxury hotels and their prices, we determined additional charges for these busy dates. For example, if a guest books a room on the weekend, we add $100 per night to the fee. If the guest books a room in a "busy" month, i.e. December, March, June, July or August, we add an extra $150 dollars to the price.

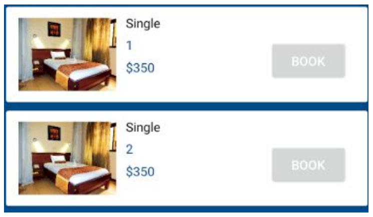
Normal Base Price:



Single
1
$250

Single
2
$250

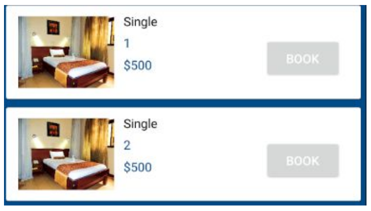
Weekend Price:



Single
1
$350

Single
2
$350

Weekend Price in July:



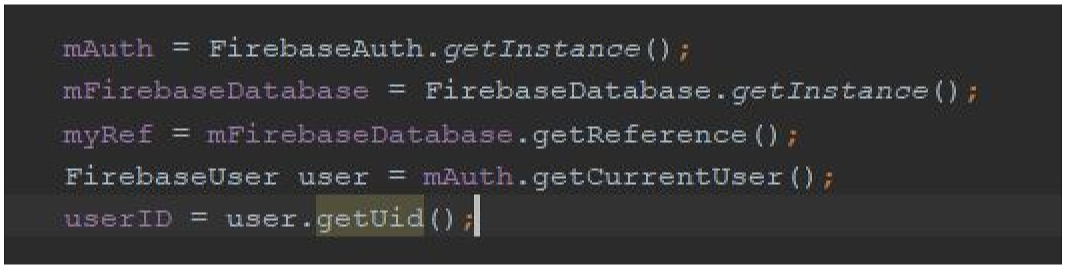
Single
1
$500

Single
2
$500

Data Structures

The place where we use complex data structures is within the data storage. We upload the data about all the Users of the application into a Google Firebase. Firebase is a NoSQL database and stores its data in JSON format. The database is essentially one large dictionary object (similar to a Hash Table) with various different dictionary objects nested within it. Thus, whenever we need to access we need to pull up information regarding a specific user we can use a key value, which would be a userID, to get the value, which would be all the information about that user stored in another dictionary. You would access the parts of the user information dictionary just as you would access the previous dictionary of all the users. We had to work with this dictionary structure due to the way Google firebase operates. Since Firebase is very easy to integrate into Android applications, We decided it would make sense to use within our app. Thus we had to think about how we would represent our users within a dictionary. Thus, it was helpful to think of our users as objects with attributes that could be represented with the dictionary. An example of how we stored our data can be viewed below:

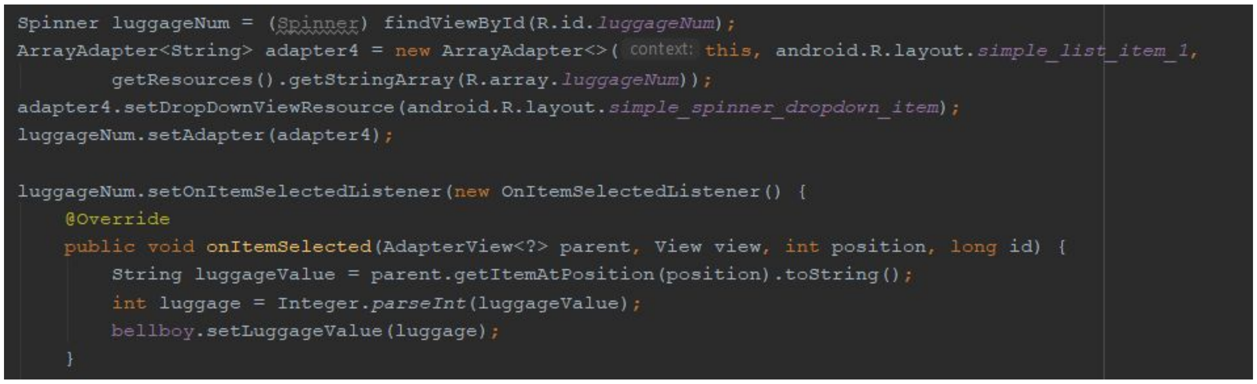
Looking at the bellboy service activity, a type of this service request, we used the following code to pass our data into Google Firebase:
First, we initialized the Firebase connection:



```
mAuth = FirebaseAuth.getInstance();
mFirebaseDatabase = FirebaseDatabase.getInstance();
myRef = mFirebaseDatabase.getReference();
FirebaseUser user = mAuth.getCurrentUser();
userID = user.getUid();
```

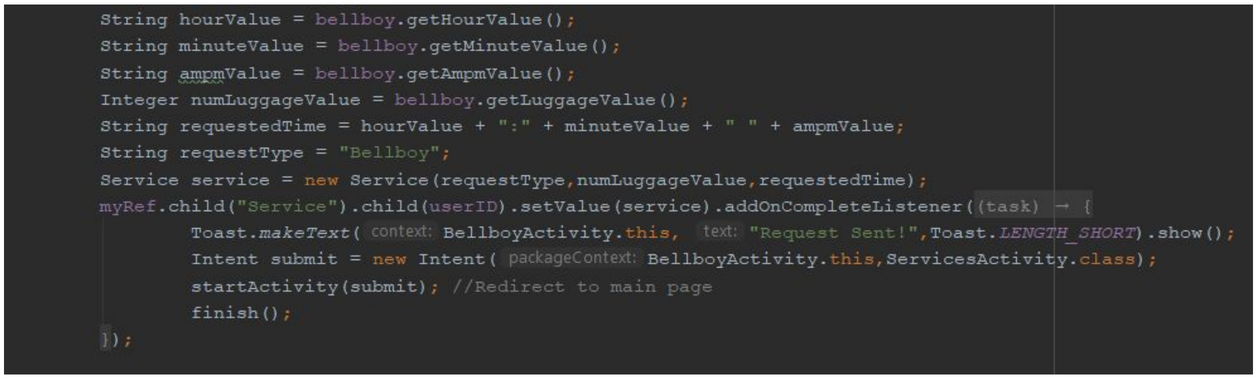
After this, we used the following methods to pass user inputted values into our Services Object, whose attributes and methods can be found in the listed in the classes section above.

An example here is storing the value collected by a spinner widget to track the number of luggages a guest is bringing, then passing in the recorded value into a setter method from the Services Object.



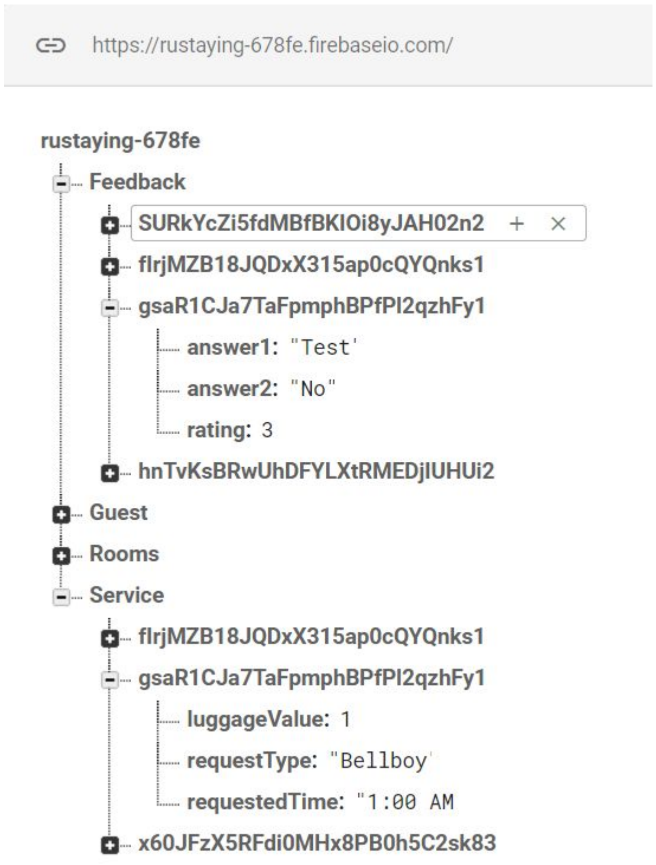
```
Spinner luggageNum = (Spinner) findViewById(R.id.luggageNum);
ArrayAdapter<String> adapter4 = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1,
        getResources().getStringArray(R.array.luggageNum));
adapter4.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
luggageNum.setAdapter(adapter4);

luggageNum.setOnItemSelectedListener(new OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        String luggageValue = parent.getItemAtPosition(position).toString();
        int luggage = Integer.parseInt(luggageValue);
        bellboy.setLuggageValue(luggage);
    }
```

This same process is repeated for each user inputted value on the service page. Eventually, we use getter methods to get the values we just set previously, and then create a new JSON object to pass into the database:



```
        String hourValue = bellboy.getHourValue();
        String minuteValue = bellboy.getMinuteValue();
        String ampmValue = bellboy.getAmpmValue();
        Integer numLuggageValue = bellboy.getLuggageValue();
        String requestedTime = hourValue + ":" + minuteValue + " " + ampmValue;
        String requestType = "Bellboy";
        Service service = new Service(requestType,numLuggageValue,requestedTime);
        myRef.child("Service").child(userID).setValue(service).addOnCompleteListener((task) → {
                Toast.makeText( context: BellboyActivity.this,  text: "Request Sent!",Toast.LENGTH_SHORT).show();
                Intent submit = new Intent( packageContext: BellboyActivity.this,ServicesActivity.class);
                startActivity(submit); //Redirect to main page
                finish();
        });
```

Now, this data should be stored in real time in Google Firebase. An example of viewing our guests' data in Google Firebase itself online can be found in the screenshot below:



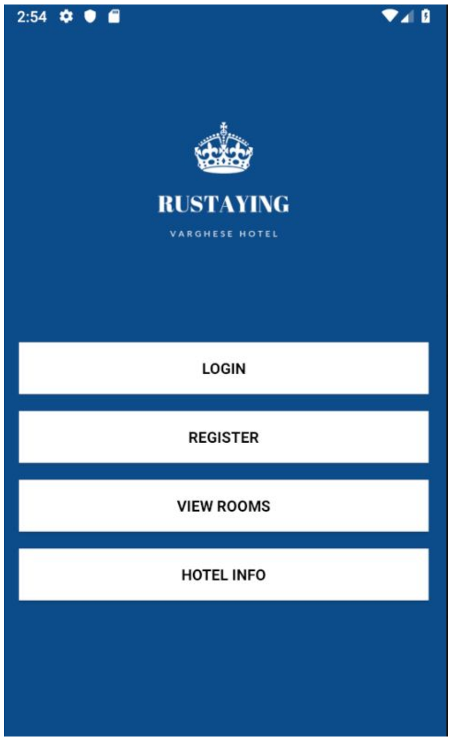
```
https://rustaying-678fe.firebaseio.com/

rustaying-678fe
  Feedback
    SURkYcZi5fdMBfBKIOi8yJAH02n2
    flrjMZB18JQDxX315ap0cQYQnks1
    gsaR1CJa7TaFpmphBPfPl2qzhFy1
      answer1: "Test'
      answer2: "No"
      rating: 3
    hnTvKsBRwUhDFYLXtRMEDjlUHUi2
  Guest
  Rooms
  Service
    flrjMZB18JQDxX315ap0cQYQnks1
    gsaR1CJa7TaFpmphBPfPl2qzhFy1
      luggageValue: 1
      requestType: "Bellboy'
      requestedTime: "1:00 AM
    x60JFzX5RFdi0MHx8PB0h5C2sk83
```

Viewing this as a set of directories, we can see on the left most column, we have Feedback, Guest, Rooms, and Service listed, which are 4 examples of data that we storing. From here, in the feedback, we see 3 users with randomly and automatically generated user ID's. Clicking on the 3rd guest, we see his answers to our feedback form, having been stored in real time into the database. From here, we can also see the service directory, and look at the second guest. A quick inspection of the user ID will validate that this is the same guest whose feedback we just viewed. We can then also see his answers to our service request, and that this data has also been stored.

Chapter 10

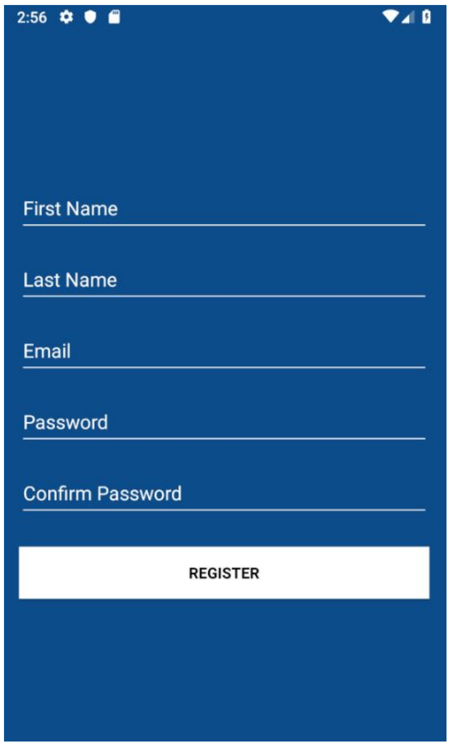
User Interface Design and Implementation

Implementation of our app was built on Android Studio using Activities to display the contents of each page that the user sees. We use XML to create the contents that the user sees in each activity and Java to provide functionality for all the elements on the screen such as textboxes and buttons. The application can be run locally on an emulator built into Android Studio or you can connect an Android device and run our app. The interface is easy to use and displays information in an easy to read fashion. Every button is labeled and provides users with messages using Android Toast Widget to provide feedback when a user has completed any action successfully. The UI design is simple and on focuses on the necessary information. There are no pictures or unnecessary data to distract the user from the task they want to accomplish. Overall the design from our mockups is unchanged. We added some stylistic changes by using Material Design by Google and changing the colors, However the overall frame and design is simple and functional. Below we have a couple pages of our application:

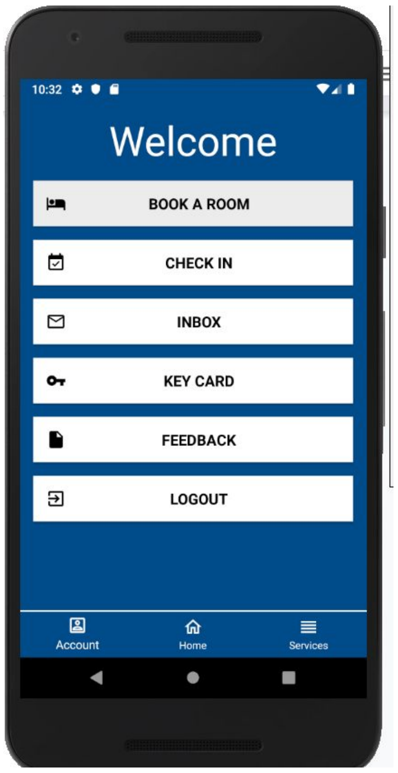


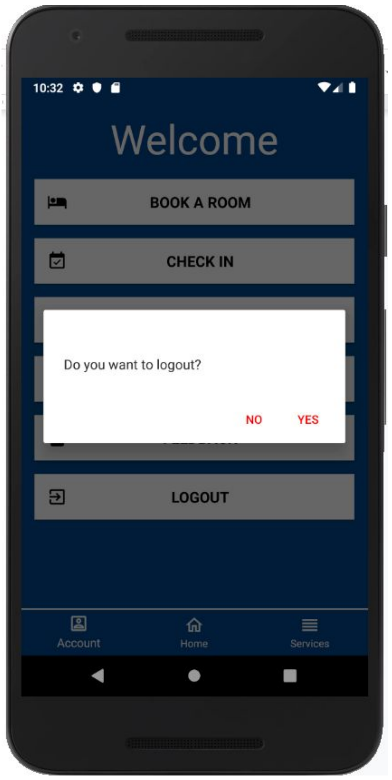
This is the main page that opens up when a user opens the app. From this page, the user can login, register to create a new account, view rooms and read more about our hotel. We kept it fairly simple and included our hotel logo as well.

The register page is also very simple with every box fully labeled so the user knows exactly what to input. We also have functionality in our code that provides popup notifications when text fields are empty so the user knows if they are doing something wrong.



After logging in the user is brought to the main page. From here the user can access all the things the app provides. The buttons are all large, clearly labeled, and have images to help the user.





We also have alert dialogs in case a user accidentally clicks a button. This is all to make sure our app is easy to use and the user is correctly deciding each action they want to do.

Chapter 11

11.1 Design of Tests

Test Case 1: Guest Login
Use Cases: UC-1, UC-2
Pass/Fail: A successful test will verify the guest login information to allow or deny access
Function Called: login(guestEmail, guestPassword)

Test Procedure:	Expected Results:
Guest enters their correct email and password	The system verifies with the database and allows access to the guest
Guest enters incorrect email or password	The system notifies guest of incorrect login and prompts to login again

Test Case 2: Guest Registration
Use Cases: UC-1
Pass/Fail: A successful test will add a new guest into the system
Function Called: addUser(guestEmail, guestPassword)

Test Procedure:	Expected Results:
Guest fills in all required fields of registration form	The system creates new account and stores it in the database
Guest already has an account that exists	The system notifies guest that their email is already in use by another account
Guest does not fill in all the required fields	The system notifies guest to fill in required fields

Test Case 3: Admin Login
Use Cases: UC-1, UC-2
Pass/Fail: A successful test will allow an admin to login with preset email address and password credentials
Function Called: adminLogin(adminLogin, adminPassword)

Test Procedure:	Expected Results:

Admin enters their correct email and password	The system verifies with the database and allows access to the admin
Admin enters incorrect email or password	The system notifies guest of incorrect login and prompts to login again

Test Case 4: Feedback Form Use Cases: UC-13 Pass/Fail: The test passes if the feedback data from the guest is properly sent and stored into the database Function Called: collectFeedbackData()	
Test Procedure:	Expected Results:
The guest fills out feedback form and data is sent to the database	System updates the database with the new data received

Test Case 5: Design Admin Pages Non-functional Test Function Requirement: REQ-18 Pass/Fail: The test is successful if the proper pages are displayed when admin logs in	
Test Procedure:	Expected Results:
Admin has logged in	The system noticies admin login and redirects to proper XML pages

Test Case 6: Display All Rooms Use Cases: UC-4, UC-7 Pass/Fail: The test passes if all the rooms available in the hotel are presented to the guest when they go to create a reservation Function Called: populateAllRooms()	
Test Procedure:	Expected Results:
Guest requests new reservation	System will populate the UI elements to show all the rooms available

Test Case 7: Room booking Use Cases: UC-4, UC-7 Pass/Fail: The test passes if the user is successfully able to book a room and filter and update the available list of rooms.

Function Called:	
Test Procedure:	Expected Results:
Guest requests new reservation and specifies room type and duration of stay	System will update the list of all available rooms to match the requirements of the guest

Test Case 8: Forgot password Use Cases: Pass/Fail: The test passes if the user is able to reset their password if forgotten. Function Called: changePassword()	
Test Procedure:	Expected Results:
User requests a password change at login page by clicking "Forgot Password"	System will update the specified user account and it's password

11.2 Unit Testing

Unit testing is software testing for each unit of a software. Unit is the smallest part of the program. For our case, the smallest component of the software is a method. Each method has one or more input and a single output. For unit testing, we decided to test six integral methods of the application. You will find those methods in the test cases listed above. Our goal is to validate that each unit or method performs as designed. For example, we are testing Guest Registration (Test case 2) to check that each user can successfully create an account. If this method does not properly perform its tasks, the user will not be able to use any further features of the app. By providing such testing, we can ensure that our code is reliable and reusable.

11.3 Test Coverage

Test coverage is the amount of testing done on a piece of software by a set of tests. It includes gathering information about parts of the code you are going to test. So, it ensures that the tests you are running are successfully testing what they are designed to test. Benefits of test coverage are that it guarantees the quality of your test and also helps identify parts of your code that are fixed or altered. It helps to create a number of various test cases to increase coverage. As mentioned under Unit Testing (3.1), we want to guarantee that our users can successfully create their accounts. Without so, they would not have further access to the app. We are providing a diverse set of test cases to ensure that the method works properly for different sets of input. By increasing the number of diverse test cases we perform, we are increasing test coverage.

11.4 Integration Testing

Integration testing is one level above unit testing that tests a group of units together. The point of such testing is to ensure units work well together and eliminate any faults in interaction of units. For example, when we test Guest Login, we check the interaction of many units coming together. We are directly testing the methods required to have a user login, as well as algorithms to store and receive user information required to login. We test the database to make sure information the user typed in is valid. When we test the Feedback Form, we are again testing its direct code as well as the database. When a guest fills out the form, we want to ensure that the data is being properly stored in the database. The system would update the database when the new information has arrived. By using integration testing, we are verifying that our methods work together to provide a functional application.

11.5 Algorithm Testing

As mentioned under 1.1 (Algorithms), our application has no real algorithms needed for any calculations or predictions. For testing non-functional requirements, we are testing REQ-18 from Report 1. The requirement states that the app should have a simple and responsive user interface. Our Test Case 5 is a non-functional test that determines if proper pages are displayed when admin logs in. The system is expected to recognize that an admin, not a guest, logged in and should redirect to the proper XML pages.

Chapter 12

History of Work

Summary of the major accomplishments thus far:

- Successfully created a simple interface for guest and admins to access the app
  - UI design schemes
  - Navigation maps
  - Other design schemes
- Database stores and verifies guest/admin account information
  - Store data collected from users so admin can see from UI
- Switched from SQLite local data to Firebase database system
- Construct key features using Firebase. These fully implemented features include:
  - Request Bellboy
  - Login/Logout for both guest and admin
  - Registration
  - Make a Reservation
  - Feedback Form
  - User Information

6.1 Milestone 1 - Improvements from Demo 1

Rank: 1
Projected due date: 02/22/2019 (Before Demo 2)

6.1.1 - Design of Application

Description: Change the design so that the user can view the rooms, availability, and prices before having to log in or create an account to book it. When viewing the rooms and their availability, also add photos, prices and what type of bed/amenities come with that specific room for the user to be able to view.

6.1.2 - Feedback Form Changes

Description: Change the feedback form to rank specific target questions with stars to take in numbers from 1-10 for feedback in order to do a data analysis to see which aspects of the hotel need improvement. These aspects include but are not limited to:

- Rate our staff
- Rate our car service
- Rate our overall cleanliness
- Rate our amenities
- Rate your overall experience

6.1.3 - Employee Login

Description: Create a login for employees so that they can view what they need to do based on their roles (manager, bell boy, etc.) and the requests made for them by the guests directly rather than having to go through the admin and then the admin make the request.

6.2 Milestone 2 - Working front-end modules for most-used pages

Rank: 1
Projected due date: 02/28/2019
Description: To have the app do the bare minimum, we have to finalize the app display so that we have a template that we can work with and maintain consistency throughout the project
The most-used pages are accordingly:

- Sign-up page
- Login page (Not logged in)
- Home page (Logged in, user accesses functions through here)
- “Make a reservation” page

6.3 Milestone 3 - Implement the combination of most-used pages in an app framework

Rank: 1
Projected due date: 03/15/2019
Description: For the most used pages (Milestone 2), the the functionality will come later down the line. Laying down this framework does not require databases but rather a working app where we can jump from one page to another seamlessly after login. This helps build our foundation to build the rest of the project on.
Combining the most-used pages in app form:

- Sign up page
- Login page (Not logged in)
- Landing page (Logged in, user accesses functions through here)
- “Make a reservation” page

6.4 Milestone 4 - Implement functionality for most-used pages

Rank:1
Projected due date: 04/22/2019
Description: To have a working app which has the capability of the most basic utility, the most-used pages need to be in performing condition. Space and efficiency is not a priority in this situation but a mock customer should be able to :

- Sign up for an account
- Log in to the app

- Reset password if user forgot and cannot login
- Make a reservation with specific dates
- Receive an email for confirmation
- Log out successfully

6.5 Milestone 5 - Implement a managerial login which has more security clearances than regular guest login

Rank:1
Projected due date: 03/20/2019
Description: Manager should be able to login and view customer interaction by selecting a customer in database and pull tables of info from it. Primarily for hotel security and app maintenance as well to make sure no requests go unnoticed/unfulfilled.

6.6 Milestone 6 - Design a page for remote check-in/out

Rank: 2
Projected due date: 03/24/2019
Description: In order to implement remote functionality for guest's convenience, we need to first create a display template for the users to choose between the different options. These options are included below.

- Check-in
- Check-out
- Digital Key

6.7 Milestone 7 - Implement remote functionality for guest's convenience

Rank: 1
Projected due date: 03/24/2019
Description: At the guest's convenience, they can check in/out and request a digital key if they would like to help make their stay more comfortable. Check in/check out application will send a request to the appropriate staff who will be ready for a guest checking in/ dropping off key. Digital key will generate a key in the backend, store it in the database and send a copy to guest.

6.8 Milestone 8 - Design pages for guest services

Rank: 2
Projected due date: 03/31/2019
Description: After guest has checked in, they should be able to utilize services that the hotel provides. Displays without functionality will be needed before we link to the app and start backend. Included as below.

- Room Service
- Maintenance Request
- Bellboy Service
- Driver/Taxi request

6.9 Milestone 9 - Implement guest services functionality

Rank: 1
Projected due date: 03/31/2019
Description: Since each service available is unique and has its own responsibilities, we have to implement a backend system which will take the data received in the app, cross check with database tables , and output accordingly.

6.10 Milestone 10 - Design pages for static FAQ

Rank: 2
Projected due date: 04/07/2019
Description: A display page is needed to showcase answers to frequently asked questions by guests and to have available a number to reach the front desk

- Concierge Service
  - FAQ
  - Phone number to call front desk
- Hotel information
  - Restaurant menu
  - Gym hours
  - Pool/Spa hours

6.11 Milestone 11 - Implement forms for requests and feedback

Rank: 2
Projected due date: 04/07/2019
Description: Static pages which have the same functionality of google forms where customers can send feedback and/or ask more questions if they so choose.

6.12 Milestone 12 - Debug and begin prepping final demo presentation

Rank: 2
Projected due date: 04/15/2019
Description: Apps will always have some mistakes which can only be identified during runtime. Our team will thoroughly test and document all possible cases of normal usage and solve any apparent errors. We will also be making a routine that the average guest would go through during their stay, this also helps with extra testing.

6.13 Milestone 13 - Final Check

Rank: 1
Projected due date: 4/20/2019
Description: Our team will perform a final check on all our features and functionality of the app. We will run all possible tests to make sure that the app is 100% functional and has no problems. This milestone will be the launch of our app.

Possible Directions for future work:

Since we started this project from scratch, we had to limit the amount of function this platform can actually support. We focussed the basics functionalities like check in, check out, feedback, registration and bellboy services.

With limited amount of time for changes and new function, we created a platform for a mobile hotel management system.

Some ideas for possible future work include:

- Implementing a time managing systems for the hotel staff to check in and check out.
- Managing services queues automatically assign service request to each respective hotel staff member.

References

Final Report	IEEE Format
SQLite - Create a Relationship https://www.quackit.com/sqlite/tutorial/create_a_relationship.cfm	*SQLite - Create a Relationship*. [Online]. Available: https://www.quackit.com/sqlite/tutorial/create_a_relationship.cfm. [Accessed: 14-Feb-2019].
Advantages of Service Oriented Architecture https://techspirited.com/advantages-disadvantages-of-service-oriented-architecture-soa	S. Takale, "Advantages and Disadvantages of Service-oriented Architecture (SOA)," *Techspirited*, 09-Apr-2018. [Online]. Available: https://techspirited.com/advantages-disadvantages-of-service-oriented-architecture-soa. [Accessed: 1-Mar-2019].
10 Common software architectural patterns https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013	723636837799177, "10 Common Software Architectural Patterns in a nutshell," *Towards Data Science*, 04-Sep-2017. [Online]. Available: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013. [Accessed: 11-Mar-2019].
Android Architecture Patterns https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6	F. Muntenescu and F. Muntenescu, "Android Architecture Patterns Part 1: Model-View-Controller," *Medium*, 01-Nov-2016. [Online]. Available: https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6. [Accessed: 5-Apr-2019].